Data Mining and Data Handling in the Field of Bioinformatics
Ph.D. Thesis

Rafael Ördög

May 5, 2010

Ph.D. school of Informatics
Dr. János Demetrovics

Information systems Ph.D. program
Dr. András Benczúr

Supervisor
Dr. Vince Grolmusz

Department of Computer Science, Eotvos University

Contents

Chapter 1

Introduction

The appearance of computers in the 20th century revolutionized scientific research. Computer-based methods in drug discovery were inevitable. Bioinformatics was born as the study of gene and amino acid sequences, but by now it includes many other fields. Our research focused on *in silico* docking methods, geometrical study of protein structures, and database cleaning. Our earlier articles have been edited and slightly extended to form this thesis.

Chapter 1 outlines our motivations and describes where and how bioinformatics can be useful in drug discovery.

Chapter 2 is dealing with *in silico* docking methods and it is based on the article [35]. First existing docking software are discussed with focus on AutoDock 3 and genetic algorithm with local search. The rest of the chapter introduces the results of the research in the scrIN-SILICO project founded by the European Union. The project's main purpose was to find drug candidates against *Mycobacterium tuberculosis*, but the approach described can be applied to any other bacterial disease. The chapter concludes by marking the limitations of global search as the weakest point of the method, and calls attention to the possibility of utilizing the special - possibly geometrical - properties of the energy function and the specific protein.

Chapter 3 tries to answer the problems stated in the second chapter, by introducing a new way of handling 3D protein structures geometrically with Delaunay decompositions. Although the idea is not entirely new, the field is vastly unexplored, and our current results are only the first steps in exploring the possibilities. In the long term the novel methods, and statistical results presented in [34, 37, 36] may prove useful in protein structure studies, and docking site identification. The chapter is based on these articles, but contains

the same statistical results in more detail. Furthermore Appendix 1, Appendix 2 and Appendix 3 contain further supplementary material that has not been published before.

Chapter 4 addresses the problem of data cleaning, which is more related to the field of data mining than to bioinformatics, but its importance in the field is undeniable. Most of the biological data banks were designed in the early stages of informatics, when automated mass processing of entries was not considered. The chapter focuses on the Protein Data Bank (PDB), which contains more than 50,000 entries, and it is a large repository of 3D protein structures used in the third chapter. After briefly discussing classical automated methods of data cleaning in the PDB, a new society-based approach is introduced. WikiPDB is a portal based on the MediaWiki software that has been presented on several conferences. Apart from handling different versions of the same PDB file, it checks for syntax errors. It also provides an API for further error checking plug-ins. The primary aim is to speed up communication between research groups performing automated processing of the data bank, and between specialists of the individual PDBs.

1.1 Motivation

Although it seemed that humankind has found an efficient cure against tuberculosis in the 1960s, by the new millennium it became a significant threat once again, due to the appearance of drug resistant TB cases. Currently an estimated one third of the human population is infected with *Mycobacterium tuberculosis* and new infections occur at a rate of about one per second, most of the cases appearing in Africa and Asia. (Note however that most infections remain latent.)

In most cases drug resistance develops during - usually intermittent - treatment. In 2006 an extensively drug-resistant tuberculosis appeared in South Africa, killing 52 of the 53 patients within 16 days [20]. It is being suggested that the resistance did not develop during treatment since the majority of the patients were not infected by TB prior to the epidemic. This example of a deadly disease becoming drug resistant within a short period of time, shows that it is vital to develop new faster methods of drug research.

While some people predict a future where treatment of each disease in each individual will be specific to both the person and the pathogen, currently drug

development is a tedious task that takes years. Bioinformatics tries to speed up this process by replacing most of the *in vitro* tests with *in silico* simulations. Although our current methods are only filtering possibilities and results need to be double-checked by *in vitro* laboratory testing, they successfully reduced the amount of work and necessary equipment for finding drug candidate molecules.

As an answer to the high number of active and latent TB cases in Africa and the ever increasing number of new cases in the Eastern states of the European Union, several research projects are being sponsored by the EU or its member countries. In two of these projects - scrIN-SILICO and TB-INTER - the PITGroup of the Eotvos University took part. The research done by our group was specific to *Mycobacterium tuberculosis* but the methods developed can be easily modified to match any other bacterial disease. Our aim was to speed up several key steps in drug research by introducing new search algorithms.

Our goals can be sorted into two main groups: short term development by tweeking existing algorithms, and long term development of novel methods. The long term goals were formulated after we have realized the limits of current methods. Our aim is to set up new paradigms that can help us in going beyond our current capabilities.

1.2 An approach to curing diseases caused by bacterial pathogens

All living cells - let be it a bacterial cell or a human cell - are a complex bio-chemical systems. Enzymes regulate a network of interdependent physical reactions and a small perturbation at a sensitive point in this system can lead to the death of the cell. The aim of drug research is to understand these processes, and find molecules that can destroy pathogenic cells by destructing the proper operation of a crucial enzyme, while not compromising the functionality of the host organism. In some cases - for example in the case of antidepressants - the aim is to restore the normal operation of defective cells, but the research methods used are fairly similar.

These reaction networks can be represented as a graph where the nodes are small organic molecules and edges are reactions leading from the substrate to the product of the reaction. Sometimes these reactions are identified by the enzyme - in most cases a protein - catalyzing the reaction. The delicate

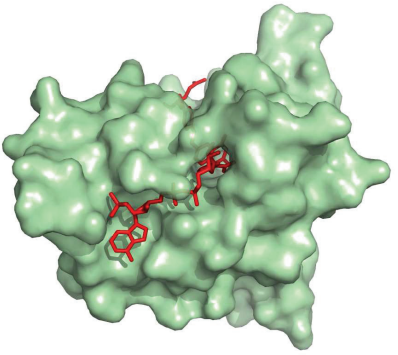


Figure 1.1: Dihydrofolate reductase complex with its substrates.

structure of these proteins allows them to speed up a specific reaction at a magnitude of even 10^7. Each living cell regulates their physical reaction networks by controlling the concentration of enzymes.

To understand how enzymes can catalyze reactions, let us examine dihydrofolate reductase. The reaction turns dihydrofolate into tetrahydrofolate. On Figure 1.1 one easily recognizes a hole on the protein surface in green. The two red molecules in that tunnel - the binding site of the protein - are the substrates of the reaction: dihydrofolate and NADPH. The enzyme binds the substrate in a way that it can present it in a conformation that is desirable for the reaction to take place easily. The part of the surface where the reaction takes place is called an active site. In the case of dihydrofolate reductase this is the same as the binding site.

Consequently by blocking the way to the binding site or the active site can obstruct the proper operation of this enzyme. This is usually achieved by finding a drug molecule which successfully competes the substrate in binding the protein at a binding or active site. Such a molecule is also called an inhibitor. In some cases the binding can be so strong that it persists for the remaining lifetime of the protein (Suicide inhibitor). After the protein is deactivated the cell eventually breaks it down and tries to replace it with another identical protein.

It is pretty clear that the destructive nature of inhibitors make this approach more suitable for killing pathogenic cells than for restoring the bio-

chemical balance of a malfunctioning cell. However it is not impossible to use these methods for finding drugs against diseases not caused by a pathogen, our main focus in this work will be to destroy cells rather than repairing them.

A drug research process against a bacteria starts by identifying the target protein that we wish to attack with our future drug molecule. This protein has to be very important to the cell, so that blocking it will be lethal for the pathogen, but has to be specific to the species, in order to minimize side effects. The classical approach to selecting target proteins is to choose proteins that are known to be important in metabolism or reproduction. Currently there are efforts in using graph algorithmic methods to search for targets in the known physical interaction network of the cell without understanding the entire system. In fact this is one of the points where bio-informatics is expected to gain key importance in the process of drug research. In this thesis we are not addressing this question, and we will expect that the target has already been identified for us.

After the target enzyme is identified one has to choose a proper inhibitor. There are available databases containing millions of molecules that can be synthesized. These databases are the result of combinatorial chemistry where molecules are produced from a limited number of building blocks. ZINC - the small molecule database we have been using - contains 13 million readily available compounds. Wet-laboratory testing all of these molecules against a target protein is infeasible. Although our current methods are insufficient for finding the best drug candidates by exclusively using *in silico* docking methods, it is possible to select a small subset of drug candidates. This smaller subset can be verified by *in vitro* testing.

Probably this is the part of the drug discovery process where bioinformatics can do the most. There are various docking methods that can predict protein-ligand docking energies. Once a small set of candidates are generated they can be tested *in vitro* against the protein. After that the best 2 or 3 candidates are chosen for further research and libraries of similar small molecules are generated. These relatively small libraries are then re-tested against the target protein with more accurate docking methods. (For example random algorithms become more reliable when they are being run repeatedly with different seeds.) In the meanwhile best hits are examined for other important properties like solvability. During this phase there is continuous communication between chemists responsible for *in vitro* testing and the bioinformatics

specialists. The very best hits are also docked against the most important human proteins to select a molecule that is less likely to be toxic or cause side effects during treatment.

The rest of the drug research process is done entirely *in vitro* and *in vivo*. There is still a long way to clinical trials and eventually to the authorization of the new drug, but we do not discuss the rest of the process in this work.

1.3 Summary of our results

As a part of the scrIN-SILICO project we aimed to speed up the docking process by making two important advances. A research group at SZTAKI clustered similar molecules in the ZINC database. Our aim was to select one representing molecule from each cluster, and only dock that single molecule. Our role was to increase the reliability of the docking algorithm, so that the number of false negative results is reduced significantly, while also reducing (or at least not increasing) the necessary number of evaluations.

First we examined one of the most widely known docking software with acquirable source code: AutoDock 3.05 of the Scripps Research Institute [10]. AutoDock uses a scoring function derived from the estimated docking energy of the ligand to the protein, and the algorithm looks for the minimum of this function. Our primary aim was to speed up the convergence of the GA-LS algorithm implemented in the software. Here GA stands for Genetic Algorithm - a heuristic global search method - and LS stands for Local Search. The algorithm tries to combine the global strategy of genetic algorithms with local search to find the optimal points from each region.

We first stated that our aim is not to find the optimum, but to generate a relevant order of the ligands. For that it would be enough if the deviation of runs with different seeds decreased with the length of a GA-LS run. We will see, that such consistency is not to be expected with this algorithm. First we showed that with the original implementation deviation can even increase in some cases and the order of ligands did not stabilize. The confidence intervals for 100 different seeds stabilized pretty soon but proved to be too large to produce a reliable and consistent order within the ligands.

We tried to reduce the deviation by distributing the same number of evaluations between more shorter runs instead of one long run. We have showed that the optimal number of runs is highly dependent on the protein-ligand

pair, and in most cases it is one or two even after 2.000.000 evaluations. Next we started to examine the relation between the global and local strategies and the results turned our attention toward the initialization process. Although one would expect that the initial population only has an effect on the first few steps of the GA-LS algorithm, it turned out that a proper choice can largely affect the speed of convergence.

Over all we have concluded that - although we could slightly improve the convergence by the choice of initial population - the GA-LS algorithm is not sufficiently reliable to find all the good drug candidate molecules against a protein, but sufficient to find at least one good molecule from each similarity group. This is not surprising though, as the no free lunch theorem [32] states, that for each global search method there exists a continuous function on which the global search method will only find a point that is much larger than the real optimum (I.e. for each $0 < k$ we can choose an f continuous function for which the real optimum is $f(y)$ and for the smallest value $f(x)$ found by the algorithm $f(x) > f(y) + k$). Consequently to speed up docking we have to understand more about the geometry of the energy function (or the protein defining it) and utilize the gained information somehow in our new global strategy.

To understand the underlying geometry of protein-ligand interactions, we have examined over 5700 "good quality" protein-ligand complexes. Our main tool was Delaunay decomposition. We examined the protein as a set of points defined by the center of the heavy (non hydrogen) atoms, and generated the Delaunay decomposition of this point set with the QHull algorithm. If our point set was in general position all regions would be tetrahedra. Unfortunately due to benzene rings, this can not be assumed. Due to numerical errors these non tetrahedral regions are being subdivided into tetrahedral regions in our current database, and we are searching for these regions in an on going research. As a curiosity we mention here, that we have found regularly appearing 5 co-spherical points in alpha helices. This structural regularity has not been reported earlier.

Before we started to search for the non tetrahedral regions we wanted to examine some of the statistical properties of the resulting tetrahedra. For visualizing we have implemented a PyMol plug-in called PyDet. First we have examined the relation between the volume and tetrahedrality of tetrahedra. We have generated a density plot seen on Figure 3.6 that showed a fine struc-

ture. The fine structures in this plot indicate that a suitable classification of the simplices may decompose it into a sum of simpler plots.

To find the classification of the tetrahedra that can correctly decompose the distribution on Figure 3.6 we have tried to group the tetrahedra by different properties. First we examined bond graphs.

Definition: The *bond graph* of a tetrahedron in a Delaunay decomposition of a protein is defined as the graph, with vertices that correspond to the vertices of the Delaunay tetrahedron, and the edges are the atom pairs closer than their van der Waals bond distance.

Similarly *covalent bond graph* of a tetrahedron in a Delaunay decomposition of a protein is defined as the graph, with vertices that correspond to the vertices of the Delaunay tetrahedron, and the edges are the atom pairs closer than their covalent bond distance.

First we examined the frequency of bond graphs. We have concluded that only 6 of the possible graphs on 4 points appear as covalent bond graphs, and all possible graphs appear as (non covalent) bond graphs. Other similar regularities can be discovered in Table 3.1, and they are discussed in detail in section 3.2.1. After we have examined basic properties of the bond graphs we have decomposed the volume-tetrahedrality distribution by bond graphs, and we have showed that there is an obvious connection between these geometrical values and the bond graph appearing on the edges of the tetrahedron.

Our second classification was based on the corner atoms of the tetrahedra. All tetrahedra were assigned a *vertex set* label that is the merging of the 4 symbols assigned with the elements in the corners in alphabetic order. Splitting the distribution plot according to the composition of the vertex-sets of the Delaunay tetrahedra would show different patterns for different labels. As an example for large differences between shape/volume distributions compare the plots on Figure 1.2.

Finally, we analyzed the atomic environments of ligand atoms, bound to proteins. Traditionally binding sites are defined by selecting the atoms in the protein closer to one of the ligand atoms than a pre-defined threshold. The choice of this threshold is not standardized, and largely influences the exact set of atoms belonging to the binding site. Our definition for atomic environment uses the notion of Delaunay regions, and requires no parameters making it more standardized. The atomic environment of each ligand atom will be identified as the vertices of a tetrahedron in a tetrahedral decomposition of the heavy

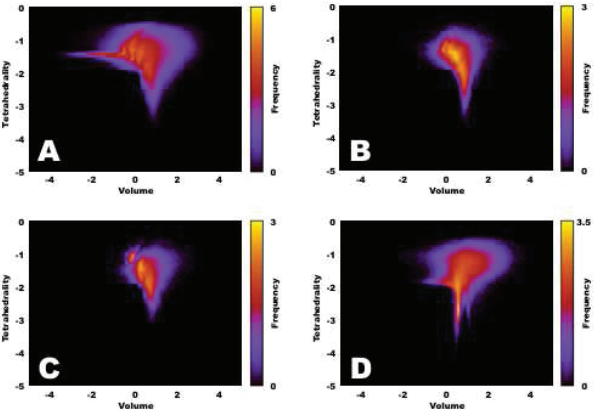


Figure 1.2: We give here similar density maps as in Figure 3.6, but now separately drawn for tetrahedra with vertices C_C_N_O (inset A), C_C_O_S (inset B), C_N_O_S (inset C) and N_N_O_O (inset D). It is clear that different vertex-compositions imply different shape/volume distributions.

atoms of the protein, containing the atom of the bound ligand.

To examine the relation between the type of elements appearing in ligand molecules, and the tetrahedron containing them, we have listed the frequency of each element appearing in tetrahedra with a certain vertex set. As it can be seen in Table 3.3 and 3.4 there are non trivial relations between the elements appearing in the ligands, and the tetrahedron containing them.

Our aim with introducing Delaunay decompositions of proteins was to understand the geometry of protein-ligand interactions better. However this work - just like any other research based on data mining the Protein Data Bank - is hindered by many shortcomings of the PDB format. In 1971 - when Meyer and Hamilton founded PDB - the primary objective was to provide a depository of structures for scientists investigating a small number of proteins at a time. That resulted in a file format and depositing rules, that is not designed for automated mass processing of the files, and it is very vulnerable to even tiny errors. The inconsistencies have various sources, from simple human errors, oversight, misinterpretation of PDB file format, to deliberately erroneous syntax to overcome certain limitations of the PDB depositing rules.

As a large number of these errors are easy to detect even with simple Perl scripts, there has been a number of attempts to correct these errors with com-

puters. Unfortunately every such approach has its shortcomings, and usually ends up in converting easy to detect syntax errors into hard to detect semantic errors.

With WikiPDB (`http://wikipdb.org/`) we have taken a different approach: a Wikipedia-like interface is provided (an extension of the Mediawiki software) for modification of the data bank files, and discussion between scientists doing research on the same structures. Our system detects syntax errors, provides an API for semantic error checking plug-ins, and keeps track of different versions. We have implemented the WikiPDB and two proof of concept plug-ins: a syntax error checking plug-in, and a bond length checking plug-in.

Chapter 2

Protein-ligand docking methods

In silico protein-ligand docking methods are becoming more and more important in searching for new drug candidate molecules because of their speed, economy and increasing reliability. Acquiring one compound (or ligand) for wet-laboratory testing from compound manufacturers costs around $ 100, consequently, without counting the costs of labor, the additional reagents and the protein production, *in vitro* verifying of the binding of one million compounds against one protein may cost $ 100 million. *In silico* simulation of the binding by docking methods costs only a small fraction of that amount, and can be completed in 1-2 weeks on a computer cluster of moderate size. The key ingredient of the *in silico* docking is the docking algorithm. Each docking algorithm optimizes some scoring function for finding the best location and conformation of the ligand molecule near to the surface of the protein. As an input, one must use the three-dimensional structure of the protein (usually taken from the Protein Data Bank) and the ligand molecule. As the output, a docking algorithm returns one or more docked conformation of the ligand and the protein, and the corresponding values of the scoring function.

2.1 Basic concepts

2.1.1 Flexibility and rigidity

Most of the docking software today handle the protein as a rigid body and allow the ligand having several torsion axes. Some software also allow flexibility to side-chains of the protein. It is easy to see that the three-dimensional position of any rigid molecule can be described by 6 real variables (three Euler angles

plus the position of one of its atoms). If ℓ torsion axes are also allowed, then each torsion axes increase the dimension by one. This means that the scoring function should be optimized in a real space of dimension $6 + \ell$.

2.1.2 The AutoDock Docking Software

One of the most widely known docking software with acquirable source code is the AutoDock 3.05 of the Scripps Research Institute [10]. Note, that the source code of such popular docking software as Dock[7], Gold[30], Fred[28], FlexX[24] and many others are not available at all.

In AutoDock 3.05 the scoring function is the estimated docking energy of the ligand to the protein. The best docking is the one with the smallest energy. The derivation of the empirical binding free energy function is described in [10], along with a brief historical review of the topic. The energy function to be optimized is a sum of the form:

$$\begin{aligned}\Delta G &= \Delta G_{vdw} \sum_{i,j} \left(\frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^{6}} \right) + \Delta G_{hbond} \sum_{i,j} E(t) \left(\frac{C_{ij}}{r_{ij}^{12}} - \frac{D_{ij}}{r_{ij}^{10}} \right) + \\ &+ \Delta G_{elec} \sum_{i,j} \frac{q_i q_j}{\varepsilon(r_{ij}) r_{ij}} + \Delta G_{tor} N_{tor} + \Delta G_{sol} \sum_{i,j} (S_i V_j + S_j V_i) e^{-\frac{r_{ij}^2}{2\sigma^2}}\end{aligned}$$

with same notations as in [10]. To speed up evaluation of the energy function AutoDock uses a grid-based approach. First, the protein is preprocessed by a program called AutoGrid: probe atoms and charges are placed at grid points around the protein, and the energy function is calculated and stored for latter use. After that AutoDock uses trilinear interpolation between grid-points to determine energy terms for each atom of the ligand separately, and then sums them up to calculate the energy of the conformation.

The strategy is to minimize the energy function above in the $6 + \ell$ dimensional real space where each point of the space corresponds to a conformation of the ligand. The optimization procedure clearly distinguishes a *local optimization* and a *global optimization* strategy.

The aim of the local minimization strategies (local search) is to find a local minimum of the function in the neighborhood of the starting point.

The aim of the global minimization strategy is to find the minimum of the function on the whole domain.

We need to mention that although there exist a good number of local search

techniques finding local optima reliably for any continuous function, the *No free lunch theorem* [32] states that it is impossible to find a general purpose algorithm for global optimization, that performs equally well on all functions. Hence it is not trivial - if possible at all - to choose a global strategy, that suits a class of functions well. In order to circumvent this problem, one needs to utilize as much information about the function as possible.

The first versions of AutoDock were using Simulated Annealing (SA) as global optimization strategy [10]. Further investigations [12] showed that Genetic Algorithms with Local Search (GA-LS) - discussed in the next section - outperform the SA strategy.

AutoDock Version 3 and latter have both SA and GA-LS implemented. Neither for SA nor for GA-LS convergence results are known in the case of the energy function above. Note also, that one can rarely prove convergence theorems for global optimizing algorithms, and even if it is possible to prove convergence - like for graph bi-sectioning with SA [14] - the proof is very complicated.

Simulated Annealing starts with a randomly chosen point in the search space, and repeatedly tries to improve the solution, by replacing it with a randomly modified one. In Simulated Annealing, only certain perturbations that increase the function value, but all that decrease it, are accepted. For more information on SA, see [14].

2.1.3 Genetic Algorithms

This section is a brief summary of notions in the field of Evolutionary and Genetic algorithms, and may be skipped by the experienced reader.

Algorithms - performing function optimization - based on the principles of Darwinian Evolution, are called Evolutionary Algorithms. These algorithms maintain a collection of possible solutions (individuals) and select certain individuals for further processing depending on their fitness, i.e., the function value at the point represented by the individual. The most widely used of these algorithms are the Genetic Algorithms. In this section we sketch the basic properties of Genetic Algorithms. A more detailed description is given by Whitley [31].

An *Individual* is a point in the search space, and its *Genotype* is the string of numbers (or vector) that describes it. The *Phenotype* is the collection of attributes of the individual, and its *Fitness* is the function value corresponding

to the individual. A *Population* is simply a collection of individuals. In our case Genotype is the set of coordinates of an atom in the molecule, and the torsion angles, Phenotype is the corresponding conformation, and Fitness is the value of the energy function calculated for the given conformation.

The algorithm first selects a population of (usually random) individuals that form the first *Generation*, then enters a cycle of deriving the n^{th} generation of individuals from the preceding. Every generation has the same fixed size. A cycle of the algorithm performs a *selection* and applies *variation operators* to the individuals in the current population.

We perform a *selection* by randomly choosing each individual of the new generation from the current generation. We expect the selection operator to keep at least the best individuals alive with high probability. Either one evaluates all of the solutions and make a selection after this, or one may review just a subset of solutions. Popular methods include: selecting deterministically the best individuals, performing roulette wheel selection proportional to the fitness of each individual (*proportional selection*), or *tournament selection*, where one chooses a number of random individuals and keeps only the best few. *Binary tournament selection* is done when we keep the better of two random individuals. We talk about a *probabilistic tournament selection*, if we perform proportional selection on a randomly chosen subset of individuals.

A *variation operator* is a randomized procedure deriving a fixed number of individuals from the same number of individuals. The two main groups of variation operators used are *mutation* (unary) and *crossover* (binary) operators. It is anticipated in the case of mutation that a good solution might be improved by a small perturbation, and also this kind of operation helps keeping the population diverse. When one performs a crossover it is expected that the good properties of two different individuals will be combined in an offspring.

Mutation operators are usually replacing the original individual by another one, by adding a random vector to the genotype, where the distribution is usually concentrated near the origin. In the case of continuous optimization the typical choice of distribution is the Cauchy or the Gaussian distribution with the variance either fixed or changed over generations according to an *annealing scheme*.

Commonly used crossover operators are either *combinatorial* or *arithmetical*. A combinatorial operator would choose a set of random positions where

bits are taken from one of the individuals and all other bits are taken from the other. More specifically a *one point crossover* operator would choose a random border position and copy bits before this position from the first individual, and the rest from the other, while *two point crossover* is copying bits between two random border positions from one individual, and the rest from the other. We can further classify one and two point crossover operators according to the distribution used to choose the border positions. A *uniform crossover* chooses uniformly from all possible positions, while with *block crossover* we only choose uniformly from a subset of allowed positions that separate certain blocks. With the latter the idea is that, splitting certain bits belonging to one chunk of data into parts may result in the offsprings phenotype have scarce in common with the phenotype of the ancestors, hence we can not expect it to combine the virtues of them.

When performing continuous optimization we can further use *arithmetic crossover*, where offsprings are generated as a weighted mean of the ancestors, i.e., the i^{th} coordinate of the offspring is of the form $c_i = r_i v_i + (1 - r_i) w_i$, where v_i and w_i are corresponding coordinates of the ancestors, and r_i is a randomly chosen value between 0 and 1. We are talking about a 1-weight arithmetic crossover if all the weights are chosen to be the same, and n-weight crossover, if we choose weights for the coordinates independently. Some implementations also maintain a list of best discovered individuals that may never be excluded from a new generation. With this latter technique called *elitism*, it is expected to exploit solutions similar to a discovered optimum. The process of deriving generations is continued until a *termination condition*. A trivial condition is to stop when the optimal solution has converged to a certain extent, but this is rarely satisfactory. The most popular condition is maximizing the number of function evaluations, or generations, or both of them.

Genetic Algorithms with Local Search (GA-LS) introduce a further unary variation operator that is usually not considered a mutation operator and are typically applied at the end of a cycle. This variation operator is a type of *Local Search*, typically Solis & Wets random local search. (See below.) An extensive work on global optimization with GA-LS algorithms is reviewed in Hart's thesis [12].

The Solis & Wets random local search [25] adds a random derivate to each of the coordinates of the genotype in every step, and if the new individual has

1. Generating Individuals
2. Loop through generations
   (a) Global strategy (Genetic Algorithm)
      i. Selection
      ii. Crossover
      iii. Mutation
      iv. Elitism
   (b) Local Strategy (pseudo-Solis & Wets local search)

Figure 2.1: Structure of a GA-LS implementation

better fitness value then it continues from this point, otherwise the previous one is repeated with the negative of the derivate. If neither of the changed individuals are accepted, we move on to the next step with the same individual. The step size is varied according to the number of successful and failing steps. More exactly, it is allowed that the search takes larger steps when going downhill, and smaller ones when trying to locate the exact position of optima already approximated.

The GA-LS implementation in AutoDock

To initialize the first generation, AutoDock chooses a population randomly. Population size is a parameter that is set to 50 as default. Then it enters the generation-deriving process, and starts the global strategy. Selection is either proportional or a probabilistic binary tournament. We applied proportional choices for our experiments. Before the crossover step, individuals are being permuted, and then subsequent $2i^{th}$ and $(2i+1)^{th}$ individuals are being applied the crossover operator with a given probability, that is $\frac{4}{5}$ by default. AutoDock is using either one or two point block crossover (latter being our choice) where the blocks are real values for translation, rotation, and torsions. Cauchy mutation is being applied with default parameters mean 0, variance 1, and probability of applying the mutation operator is $\frac{1}{50}$. Elitism was set to keep the best individual alive. As a default every individual of a generation is applied a local search with the probability $\frac{2}{3}$, maximal number of steps is 300. All further parameters are left as default in AutoDock.

2.1.4 Previous Work

Thomasen's remarkable paper [29] analyzes genetic algorithm parameterization in the AutoDock 3.05 software. Six protein-ligand complexes ([29], Table 1) from PDB entries 3ptb, 2cpp, 2mcp, 1stp, 1hvr, 4hmg were chosen for the experiments, and the effect of different population sizes, mutation operators, recombination operators, different local search strategies were tested for these six complexes. Based on the findings, in [29] a new algorithm, called DockEA was introduced and compared with the original AutoDock solutions.

The results were quite different for different complexes: for some of the complexes it was not too difficult to find the near-optimum solutions, and for some others it was a more challenging task. Note also, that the resolution of the complexes examined was in the range 1.63-3.1 Angstrom. The termination criterion was set to 250,000 evaluations and each experiments were repeated with 30 random seeds.

We, in contrast, docked 48 randomly chosen ligands to the same protein, and each docking was repeated 100 times with different random seeds. The main point of our analysis was to find the number of evaluations which already gives reliable results, but it is as low as possible to facilitate screening large ligand libraries. Consequently, we perform our experiments with higher evaluations upper bound than 250,000; in some cases the upper bound was set to 2 million.

2.1.5 Concept of our analysis

Clearly, an "idealistic" randomized algorithm that performs automated docking needs to satisfy some natural requirements. The nature of requirements may depend on the actual problem we are dealing with. Our main motive was to optimize a database screening task with a single protein against two million ligand molecules without changing the energy function of AutoDock. In what follows we list our requirements and give methods to quantitatively evaluate them.

Capability of finding optimum: This requirement means that with high enough probability (i.e., for lots of random seeds) the algorithm approximates the minimum of the function with a small enough error. (In our case it would be enough if the calculated energies were precise enough to give a "relevant order" described below.) Evaluating this requirement is difficult, since we do

not have reliable measurement data on the values of the energy-minima for the ligands in question. On the other hand, if we had such data measured by biochemists in wet-lab experiments, then a negative result could still mean that the used energy function is invalid, and not the minimization procedure. To circumvent these difficulties, we compared different variants of the GA algorithm based on a fixed number of runs each.

Mean approximates best run - (Low deviation) It is reasonable to expect that a result, yield by a given run of the algorithm is at least generally close to the optimum. This requirement is important since otherwise every docking experiment would have been repeated many times to get usable results. The requirement can be measured by variance and (standard) deviation.

Consistency of the above quantities would mean that if we increase the number of evaluations, then the discovered optimum gets ever closer to the real optimum and deviation drops as well.

Relevant order If one is searching for ligands that bind to a certain protein then an order of ligands with respect to their docking energies would be helpful, even without the actual energies. One could then proceed by increasing accuracy with more time consuming methods for the first few hits. Consequently an algorithm that determines this order (or approximates it in some sense) - even if it miscalculates minimal energies, or it doesn't even attempt to calculate them - could mean a first step in finding possible drug candidates. We settled for consistency in this respect, that is, if the number of evaluations is being increased this order should stabilize, and should not depend too far on the seeds used for the random number generator.

To see the progress made by different algorithms we created run logs similar to the ones in Hart's thesis [12]. That is after every evaluation we checked if a new optimum was discovered, and recorded the number of evaluations and the new optima. On plots like the one on Figure 2.2 we can see number of evaluations on the horizontal axes, and the approximation of minimal energy in kcal/mol calculated after the given number of evaluations by different runs of the algorithm. Note that the most time-consuming step is the evaluation of the energy function, so the number of evaluations is proportional to the running time of the algorithm, independently of the hardware.

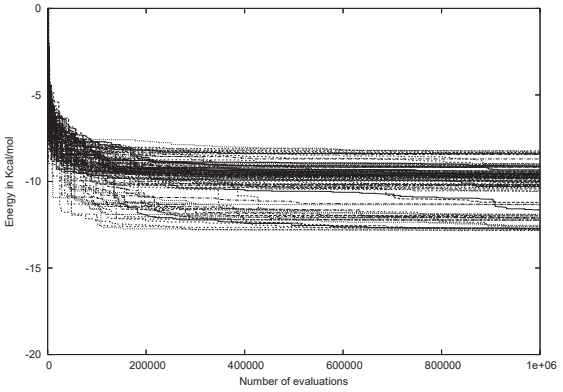


Figure 2.2: Evolution of discovered minima over 100 runs with different seeds for ligand with ZINC code ZINC01208228.

2.2 Analysis of the original algorithm used by AutoDock 3.05

Our test runs were performed on a cluster of 48 PC's equipped with 2.40GHz Intel Celeron processors and 256 MB RAM. We have randomly chosen 48 molecules from the ZINC database [15] and docked them to the same *Mycobacterium tuberculosis dUTPase molecule* as a (rigid) protein target. We were using box of $16,6x21,75x30,75$Åaround the active site with a grid resolution of $0,275$Å. The algorithm ran with 100 different random seeds for every ligand to determine deviation and seed dependency. The diagrams derived from the run logs by the procedure described at the end of the previous section were similar to Figure 2.2, and in what follows we will describe the common properties we have found:

1. In the beginning the algorithm comes across unreasonably high positive values due to the collisions, but in some 1000 steps it will reach negative values.

2. The best of 100 runs will be close to the best discovered minimum (over all runs) shortly after reaching negative values. For example on Figure 2.2 the minimal run changes less then 0.5 kcal/mol after 250 000 steps. For the 71% of the 48 ligands we were testing this changed less than

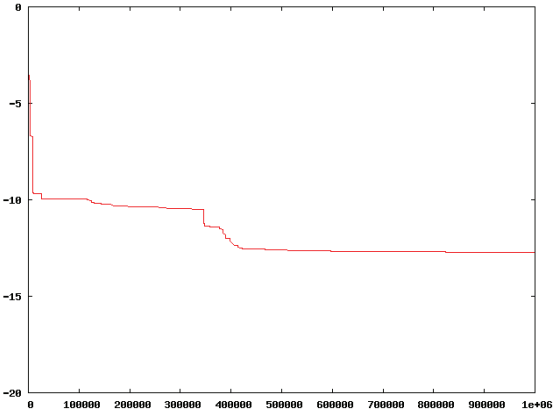


Figure 2.3: Evolution of discovered minima with a seed for ligand with ZINC code ZINC01208228. Notice the sudden fall just before 350 000 evaluations.

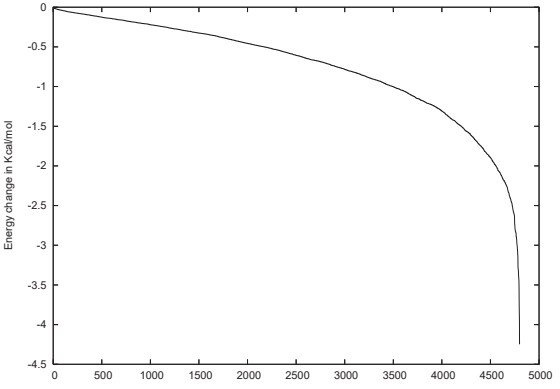


Figure 2.4: 4 800 different runs (x-axes) ordered by amount of decreasing in kcal/mol from the 250 000th evaluation to the 10 000 000th evaluation.

0.5 kcal/mol after 250 000 evaluations, and 88% changed less than 1 kcal/mol. The worst case was 2 kcal/mol.

3. As a generalization of the above observation we can see on Figure 2.2, that even after 10 000 000 evaluations most of the runs stay close to the value reached after 250 000 evaluations, hence most of them never even get near to the optimum. Note, however, that there are seeds producing large jumps at random positions. (One can easily observe such jumps on the figure just after 400 000 evaluations, where for example one of the runs descends from -10 kcal/mol to around -12 kcal/mol.)

   On Figure 2.4 one can observe 4 800 different runs ordered by amount of decreasing in kcal/mol from the 250 000. evaluation to the 10 000 000. evaluation. The 50% of all runs failed to change more than 0.5 kcal/mol, and only 25% of the runs decreased more than 1 kcal/mol.

4. Partly as a consequence of the above after 250 000 evaluations one can define a "high confidence" interval by the following properties:

   (a) Every discovered minimum is located in this interval independently of seed.

   (b) It is the smallest interval that satisfies the above.

5. High confidence intervals can be approximated by considering a small number of runs. On Figure 2.2 the maximum and minimum of 100 runs can be seen, hence vertical sections of it, can be interpreted as 100 run approximation for corresponding number of evaluations, furthermore one can also see how runs distribute in that interval. For a chosen number of evaluations one can also create plots like the one on Figure 2.6, representing the "high confidence" intervals for different ligands with vertical bars, and the mean of 100 runs with a dot on the corresponding bar.

6. Based on our 100 run approximations:

   (a) Usually (not always) results cover the interval in some sense. (One can choose an ε such that for every point one seed is within ε distance from the point and an ε is a small constant not larger than 4-10 times the length of the interval divided by the number of runs.) Our example Figure 2.2 is a counter example of this, as it has a gap

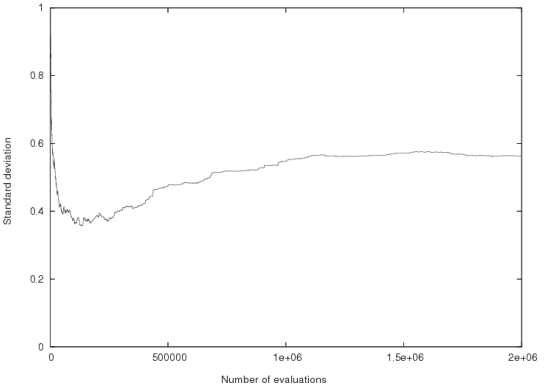


Figure 2.5: Increasing deviation of different runs on the ligand with ZINC code ZINC01106466.

between -10 and -12 kcal/mol. In our test this and another 3 ligand proved to be counter examples out of 48 ligands. The question when and why dose that happen remains open.

(b) After 250 000 evaluations this intervals' endpoints won't change much. (For the 71% of the 48 ligands we were testing the interval changed less than 0.5 kcal/mol after 250 000 evaluations, and for 88% it changed less than 1 kcal/mol. The worst case was 4 kcal/mol, but the second worst was only 1.5 kcal/mol.)

(c) The approximate size of these intervals is between 2 and 5 kcal/mol. (For example the energies for the ligand with ZINC code ZINC02958278 were between -11 and -7.)

7. Neither the standard deviation, nor the mean of 100 runs show consistent decreasing after 250 000 evaluations. (See Figure 2.5.) In fact the deviation turned out to have a slight tendency of significant increase at random positions.

The size of the minimal interval in which results are located is clearly large compared to the fact that if ligands are characterized by the minimal energies discovered over 100 different runs, the difference between the best and worst ligand was less than 6.5 kcal/mol for this 48 random ligands. (Values were

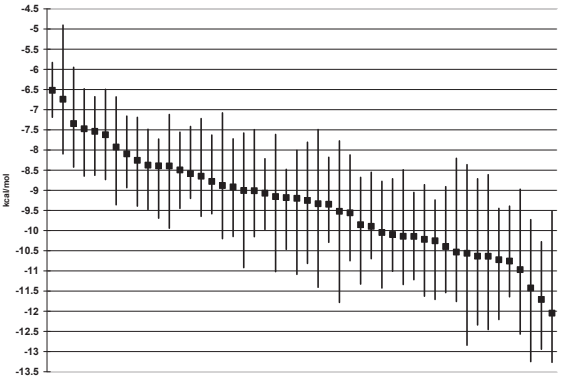


Figure 2.6: High confidence intervals for 48 ligand molecules (x-axis) after 2 000 000 evaluations.

between -13.5 and -7.) Hence a single run of the algorithm will not provide a relevant ordering of the ligands. For example one can observe on Figure 2.6 that the best ligand in terms of minimal discovered energy in 100 runs can turn out to be in the last third of the list if we use a single run strategy.

On Figure 2.6 the dots on the intervals indicate the average of runs. One can see that they are usually in the middle of the confidence interval, hence we can not expect a single randomly chosen run to be at least near to the optimum. Even if we accept the rather optimistic assumption of having chosen the "average run" for all the ligands by luck, we are still far from a relevant order, as - due to the large variance in the size of intervals - there seems to be no relationship between the minimal and average runs.

Another surprising result was that deviation sometimes was increased after a number of evaluations instead of the anticipated decreasing. (See Figure 2.5.) This pathological behavior arises when there is an easy to discover optimum, and most of the runs find it within the first few thousand steps and afterward they start to explore better solutions at random positions in time.

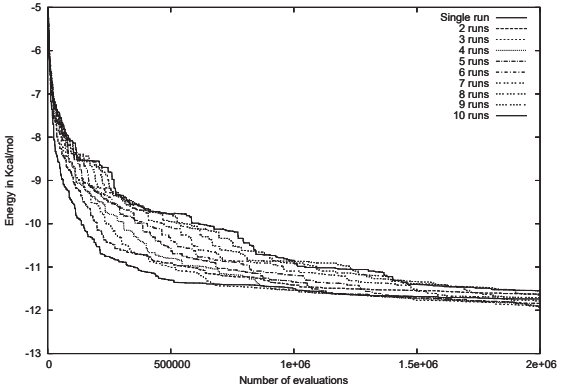


Figure 2.7: Different multi-run strategies for ZINC entry ZINC00342090.

2.3 Our results

2.3.1 Multi-run tests

In the case of high deviation it is a quite common approach to take the average, or the minimum of multiple samples. As we pointed out in the previous section averages are not relevant, consequently we tried multi-run algorithms with common minima, i.e., an n-run strategy with k evaluations would run the algorithm n times through $\frac{k}{n}$ evaluations and take the minimum of the results. Our test described below showed that optimal n for a given k is highly protein-ligand dependent, though we believe that for a group of similar ligands and fixed protein it might be possible to find a common n. If ligand classes would be large enough it might be possible to save time by preprocessing ligand classes for every protein in order to identify the optimal n. The question whether is this possible remains open.

Our tests were performed as follows: we ran the algorithm 100 times, and divided it into 10 equal sized group. For the average n run strategy the minimum of the first n out of every group was taken and after that the mean of them to get an average n run strategy.

It is obvious that for small number of evaluations any multi-run strategy is no match for the one run strategy, and for significantly large number of evaluations multi run strategies will overcome the one run strategy. (The latter is only obvious after the results of the previous section, i.e. most runs

get stuck at a point.) The question remains when dose an n run strategy overcomes an k run strategy, and more importantly which is the best for a given number of evaluations.

On Figure 2.7 one can see the average n run strategies plotted for ZINC entry ZINC00342090. In this example the 3 run strategy is the first to overcome the one run strategy at approximately 700 000 evaluations, but there is not enough difference between them to ensure that the 3 run strategy is definitely better. Another ligand showed examples where the one run strategy turned out to be far better than any other strategy even at 2 000 000 evaluations, and there were examples where the 2 and 3 run strategies overcame the single run version pretty well.

2.3.2 Modified GA-LS algorithms

The choice of the initial population is an important phase of Genetic Algorithms. Its optimization seems to be neglected by the literature. This is not surprising in a certain sense, as the algorithm usually leaves those individuals behind at a quite early stage, hence a possible bad initial choice seems to have only a negligible effect on later generations. Our investigations show that this intuition is far from being correct. In the case of a function with such high complexity as our energy function, the choice of initial population can have strong positive effect on the speed of convergence to the actual optimum by more or less restricting the search to interesting areas.

On Figure 2.8 one can observe the average of 100 runs with different initialization strategies for ZINC entry ZINC01208228. Again horizontal axes is number of evaluations, while vertical axes is energy in kcal/mol. Different strategies are described below:

Rigid start In the ZINC database [15] ligands are stored in a conformation with minimal internal energy. Hence it is a natural idea to fix the torsions when choosing the individuals randomly. Our aim was to reduce the first "many collisions" phase of the algorithm, but the diversity of the population - as we expected - turned out to be too small and that slowed down the algorithm significantly.

First population selected from a larger random population is another natural idea. To understand the behavior of random individuals we have created plots like the one on Figure 2.9 that show the energies (horizontal axes) for 5000 random individuals and the energies for the nearest local optimum found by

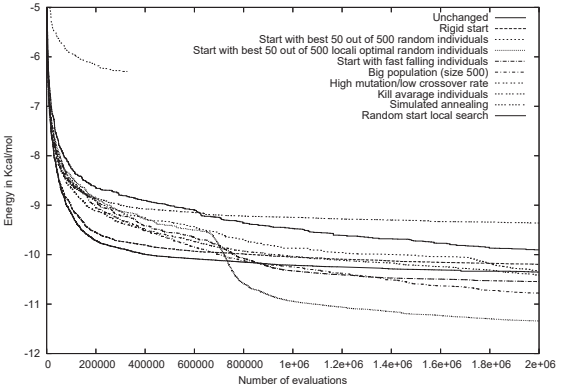


Figure 2.8: Comparing the modified algorithms (x-axis) by average run and confidence interval for ZINC entry ZINC00342090 in kcal/mol (y-axis).

local search (vertical axes). One can see that points form 3 classes, formed by low energy individuals, high energy individuals near low local optimum, and high energy individuals near high local optimum. (Sometimes one or two classes might be empty.)

Selecting the best 50 individuals out of 200000 turns out to perform more or less the same way as the unchanged algorithm. Depending on the ligand it can perform better or worse just as many times.

Selecting the best after local search with 20000 being the number of local searched random individuals and 50 the population size. As one can check on the example Figure 2.8, this approach turns out to be the best from the ones inspected by us. First it works its way through the local search phase and then soon after starting the genetic algorithm the average of the runs makes a sudden fall. At this point it outperforms all other approaches for most of the ligands.

Individuals changed by local search most dramatically seemed to be interesting, partly because they form a separate cluster (Figure 2.9) and partly because according to our notion optimal bindings are situated on the surface of the protein, hence changing it a little causes collisions, i.e., high energy conformations. This approach shared the property of making a fall after the genetic algorithm is started, but it was not large enough to outperform other algorithms discussed here.

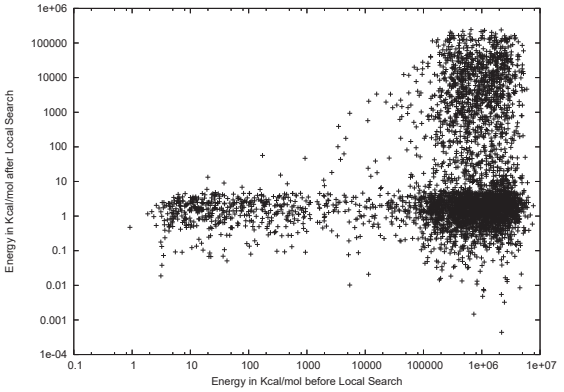


Figure 2.9: Energies of random individuals and of the nearest local optima found by local search.

Non-initialization based modifications included higher population size of 500, modified mutation $\left(\frac{4}{5}\right)$ and crossover $\left(\frac{1}{5}\right)$ rates and a modified version of proportional selection. The last modification aimed to exploit the remark mentioned earlier about optima being near to high energy conformations. We have changed the distribution so that high energy and low energy individuals had high probability of being chosen. None of the above modifications turned out to be better than the unchanged version.

Furthermore as a comparison we ran the same tests for simulated annealing and random start local search. The latter chooses a given number (5000 in our case) of individuals randomly and performs Solis&Wets local search on them with 500 being our choice of maximal number of iterations. As expected these algorithms didn't find better optima than the unchanged version. Parameters for the simulated annealing were:

- initial RT 616
- RT reduction factor/per cycle 0.95
- linear schedule
- cycles 500
- steps accepted and steps rejected equally 2000

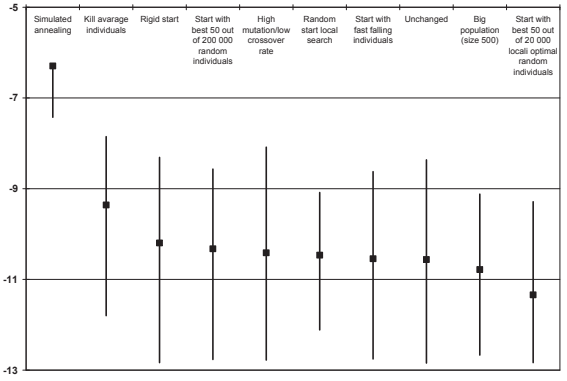


Figure 2.10: Comparing the modified algorithms by high confidence interval after 2000000 evaluations for ZINC entry ZINC00342090.

2.4 Conclusion

We have performed an in-depth analysis of the GA-LS algorithm implemented in AutoDock and its variants mostly differing in their initial populations, and concluded that they tend to have unacceptably high deviations when applied to energy functions of docking problems. Consequently, one can not expect to find exact energy bounds with this technique, nor to find a relevant order of ligands according to their bonding energies within a reasonable number of evaluations. Multi-run strategies can help, but they are highly dependent on the protein-ligand pair in question. We have stated the question if it is possible to classify ligands, and optimize the number of runs for given number of evaluations and the protein-ligand pairs in the class in a way that speeds up the algorithm.

Initialization can have a major effect on the speed of "convergence". The best algorithm examined in this article, was choosing the best 50 out of 20000 local searched individuals. The first phase of this algorithm is actually a random start local search, but after the genetic algorithm is started the average of runs makes a sudden fall and outperforms all other variants we have examined. On Figure 2.10 one can see that this variant performs better than others in average, in best and in worst runs, too.

Despite of the poor performance of the genetic algorithms on this particular problem the method still seems to work. The reason for this lies in the next

step of the drug discovery methodology. Most drug candidate molecules have undesirable features like toxicity and unsolvability in water. To overcome these problems small molecule libraries are generated for each drug candidate containing similar molecules with more desirable properties but similar binding capabilities. Since these libraries are much smaller than the entire database, it becomes feasible to test all of the potential candidates either with more time consuming *in silico* or *in vitro* methods. When using our high-throughput docking method we may miss most of the molecules that would be perfect as a final drug, but there is a high chance that in the next step we will reconsider it as part of a library generated from a similar molecule we have found.

Chapter 3

Geometrical study of 3D protein structures

In the previous chapter we have discussed a method where the initial input were finite datasets: the 3D structure of ligands and the protein. We derived a very intricate function which had to be globally optimized. An obvious drawback of the approach is that the numerical handling of continuous functions is usually considered harder than of finite spatial pointsets. In this chapter - instead of using energy functions - we will discuss methods for examining the relations between proteins and ligands directly through their structure.

The examination of straightforwardly definable discrete structures in nucleic acids and proteins turned out to be perhaps the most important development in our present knowledge and understanding of their form and function. These discrete structures are sequences of nucleotides and amino acid residues, respectively. Bioinformatics was born as the science of analyzing these sequences. The discretization of the biological information into easy-to-handle sequences of 4 or 20 symbols made possible the application of deep mathematical, combinatorial and statistical tools with enormous success. The tools, resulting from this process, changed our perception of genetics, molecular biology, and life itself.

Straightforward discrete structures can also be defined in the spatial descriptions of proteins and nucleic acids. The definition and examination of discrete objects, using the spatial structure of proteins instead of amino acid sequences would intercept spatial characteristics, that are more conservative evolutionary than the polypeptide sequences.

It is of basic importance to map the physico-chemical properties of protein-

ligand binding sites, most importantly the Coulomb and Van der Waals forces, in order to predict protein-ligand binding, to design ligands for a given binding site on the surface on a protein, or in designing inhibitors or activators in enzymatic mechanisms. The exact description of the forces in question are deep quantum-chemical problems. The atomic environment of the binding sites clearly has strong effect to these forces; consequently, by examining the atomic environments of the ligands in the crystallographically verified protein-ligand complexes in the PDB would yield insight in binding mechanisms and biologically active molecule design. The first step in this direction need to be the analysis of the simplicial structures of the atoms, forming the protein structures themselves. The second step is the analysis of simplicial neighborhoods of the ligand atoms.

Three dimensional protein structures are stored and characterized by the spatial coordinates of heavy atoms. Since our primary aim is to find the binding sites on the protein surface, it is necessary to characterize the empty spaces between the atoms. To achieve this, we will use Delaunay decompositions. We analyze the Delaunay tessellation of more than 5700 protein structures from the Protein Data Bank. The Delaunay tessellation of the heavy atoms of these protein structures give certainly a more complex structure than the polymer sequences themselves, but these tessellation are still easily manageable mathematically and statistically, and they also well describe the topological simplicial complex of the protein.

3.1 Tools of our analysis

3.1.1 Protein Data Bank and RS-PDB

The fast growing Protein Data Bank [2] is a rich depository of three-dimensional structural biochemical data today. The information stored in the Protein Data Bank would make possible fully automated *in silico* studies if misslabeled chemical groups, broken protein- and nucleic acid chains and other errors were corrected. Even today, the newly submitted data is verified "by hand" by human experts. Szabadka et.al. [27] applied a rigorous cleaning and re-structuring procedure for the entries in the Protein Data Bank, and created the RS-PDB database. We have used this cleaned database in what follows. Since the last chapter is dealing with the problem of data base

cleaning, the details of how the RS-PDB was created will be discussed later.

Our complete test set was selected from the RS-PDB by the following criteria: the entry need to contain at least one protein, with no missing atoms, and the resolution of the structure has to be at least 2.2Å. We have found 5,757 such entries in the RS-PDB database. Note, that the requirement for the missing atoms is perhaps too strict, but in this study we do not intend to deal with stability questions, i.e., the effects of the missing atoms for the whole tessellation.

3.1.2 Delaunay-Decompositions and QHull

Even the refined, cleaned RS-PDB database [27] lacks important features, such as easy acceptance of queries such as: What atoms surround a certain (ligand- or protein-) atom in the structure? Which atoms are neighboring with the atom/amino acid X in the protein? How many ligand-atoms are surrounded by exactly the tetrahedron with C-C-C-O atoms in its vertices? How frequent are the tetrahedra with vertices C-C-O-N? Are there differences in the shape of tetrahedra of different composition?

Note, that such queries cannot be answered from the amino-acid sequence of the protein, since they intrinsically depend on the tertiary structure of the protein. Consequently, one need to use some cleaned version of the PDB as the initial data.

We have chosen Delaunay decomposition in the discretization of the dataset in the RS-PDB database, since in this "tessellation", the tetrahedra are close to regular ones, and it is a natural and well defined notion, with a well-known algorithm for the generation of the tessellation.

Definition: Given a finite set of points $A \subseteq R^3$, and a $H \subseteq A$ such that the points of H are on the surface of a sphere and the sphere does not contain any further points of A, then the convex hull of H is called a Delaunay region.

Triangulations on the plane are frequently used for discretizing geometrical data and relations. Delaunay triangulations are of special importance, since they can be generated easily and the resulting triangles are usually not far from regular triangles. Delaunay regions define a partition of the convex hull of A. If the points of A are in general position, (i.e., no five of the points are on the surface of a sphere), then all regions are tetrahedra.

Singh, Tropsha and Vaisman [6] applied Delaunay decomposition to protein-structures as follows: they selected A to be the set of C_α atoms of

the protein, and analyzed the relationship between Delaunay regions volume and "tetrahedrality" and amino acid order in order to predict secondary protein structure. They gave the following definition:

Definition: [6] The tetrahedrality of the tetrahedron with edge-lengths $\ell_1, \ell_2, \ell_3, \ell_4, \ell_5, \ell_6$ is defined

$$4\left(\sum_k \ell_k\right)^2 \sum_{i<j} \frac{(\ell_i - \ell_j)^2}{15}$$

where ℓ_i is the length of edge i..

Note, that the tetrahedrality of the regular tetrahedron is 0. To find the Delaunay decomposition of a set, we have used the Qhull algorithm [5], its source is available at: http://www.qhull.org/. The underlying principle of the algorithm is the idea, that a Delaunay decomposition is easily traced back to a simple convex hull problem [8], which is then solved by an incremental algorithm.

3.1.3 PyMol

PyMol is a widely used open-source molecular visualization program maintained by DeLano Scientific LLC as a user sponsored project. Well before its version 1.0 release PyMol earned a reputation for its ease of use, and its popularity among scientist of the field. Among classic molecular models it features a powerful set of scripts for calculating physical properties of molecules, and visualizing some of them around a molecule.

The core of the program is a full featured Python interpreter, which is then extended by an OpenGL 3D display and a Tcl/Tk based GUI. This fact - apart form allowing faster development of the software - together with the PyMol API gives one the chance, of creating plug-ins. Although this feature was mainly intended for automatic coloring of atoms according to relevant properties, loading structures and manipulating structures, it is also capable of inserting additional graphical primitives, like spheres, cylinders, triangles, vertexes, called Compiled Graphic Objects (or CGO).

3.1.4 PyDeT

During our geometrical studies we felt the need to visualize tessellations derived from the protein structure, together with the source protein itself. For

that purpose we have developed appropriate PyMOL plug ins: PyDeT and PyQeVis. PyDeT has been released under a GPL license, and is described below in detail, while PyQeViz is held back regarding the fact that RS-PDB is available exclusively by members of the PIT group. Although we mention some results achieved using PyQeVis, it is possible to come to the same conclusion by a slight modification of PyDeT.

Usage

For details on how to install PyDeT please be referred to [34].

PyDeT will adopt any selection of atoms of a molecule already loaded into the PyMol software. Using the plug-in is as easy as typing the "pydet [selection][, name]" command into PyMols command line, where the two arguments - described below - are both optional.

The first argument - selection - defaults to 'all', and tells the plug-in which selection it should work with. This will take the atoms in the selection as points in 3-space, generate the Delaunay Triangulation of this point set, and visualize it by its edges. The visualization is done by creating a layer called PyDeT, and inserting 'line like' cylinders between the endpoints of the edge. The edges are colored by relative length: red corresponds to very short edges while blue corresponds to very long edges. (By relative length we mean relative to other edges in the same tessellation. Hence there will always be at least one blue and one red edge.)

Entering a name as the second argument will rename the PyDeT layer to "PyDeT_[name]", which may come handy, when one deals with tessellations corresponding to different selections.

An example

There are two different point sets that may prove to be of interest for most of the protein structures: the set of all heavy atoms (i.e. non hydrogen) and the set of Carbon alpha atoms. In an article [6] Singh, Tropsha and Vaisman examined the latter case, and concluded that Delaunay tetrahedra situated along helices tend to be smaller than tetrahedra along sheets. On the image below one can check that easily for PDB entry 1crn. Along the two helices tetrahedra inside the helix have all their edges colored red (hence they are small themselves), while other tetrahedra have at least one green, or blue

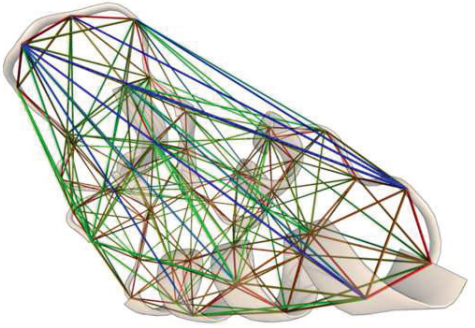


Figure 3.1: Delaunay decomposition of the C_α atoms of the protein found in the PDB entry 1crn. Delaunay tetrahedra situated along helices tend to be smaller than tetrahedra along sheets.

edge. This also indicates, that tetrahedra inside helices tend to be closer to regular tetrahedra, than other elements of the tessellation.

Structure of the python code

Pydet.py consists of 5 functions:

- **__init__** which is the initializations function. If you would like to access any of the functions from PyMol command-line, you may simply uncomment the corresponding lines.
- **PyDeT** This is a simple wrapper around function: defines some filenames that serve as an ugly way of communication with the external qdelaunay program and fires the remaining 3 functions. Still modifying this function may come handy, if you would like to add some kind of filtering on the tetrahedra before visualizing.
- **PyDeT_DExtract** Extracts spatial data from the selection, and writes it to disk.
- **PyDeT_QDelaunay** Calls qdelaunay which reads the file created in the previous step, and writes the result to disk.

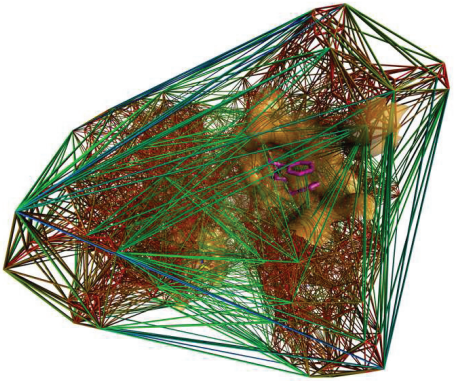


Figure 3.2: Delaunay decomposition of the heavy atoms of the protein found in the PDB entry 1crn, along with the ligand and the protein surface surrounding it.

- **PyDeT_TessVisualise** Reads the result generated by qdelaunay, and draws the edges.

We have used 3 files for communicating with qdelaunay instead of using pipes, as the pre-compiled version of PyMol dose not support piping. If one is willing to recompile pymol with the necessary module, then this ugly solution may be eliminated. (Although we are dealing with rather small files, so it is doubtful that this would worth the effort.)

If you would like to add some filter on the tetrahedra, you may want to write a function which reads all three files (and possibly also extracts further data from pymols model object) and returns the list of remaining tetrahedra. This filter function would have to go between PyDeT_QDelaunay and PyDeT_TessVisualise.

3.1.5 PyQeVis and examples

The RS-PDB was expanded by Delaunay tessellations of spacial point sets derived from the proteins non hydrogen atom coordinates, and also the point set derived from just the C_α atoms. PyQeViz extracts every data - including protein structure, and Delaunay tessellation - from the RS-PDB [27]. This enables us to filter tetrahedra by their properties stored in the database by simple

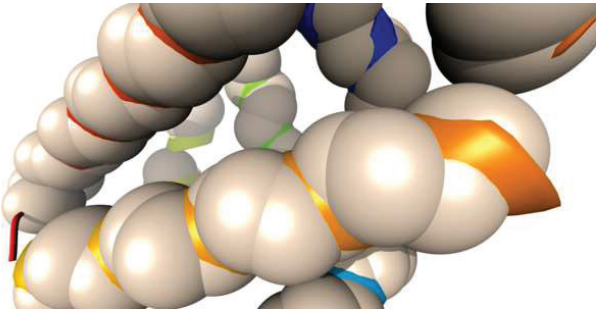


Figure 3.3: Small Delaunay spheres aligned on the helices of the protein found in the PDB entry 101m.

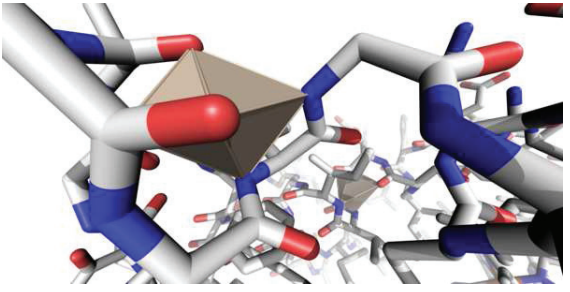


Figure 3.4: Two examples of co-spherical points in one of the α Helices of the protein found in the PDB entry 101m.

SQL 'WHERE' statements. Any of the examples below may be reproduced, by adding a filter function to PyDeT, as described in the previous section.

Let A be the set of C_α atoms of a protein. On the Figure 3.3 one can observe as the defining empty spheres with small radii (3Å at most) arrange almost perfectly to the a helices of the protein. Another interesting property of these spheres can be seen on Figure 3.4. The N of two subsequent residues, the C of the next one, and the O, and C of the fourth are frequently arranged almost perfectly. The Dealaunay tetrahedra appearing here should later be merged, and handled as one region.

3.2 Results

In contrast with the article [6], we have taken A to be the set of heavy atoms of the 5757 proteins. Note that in that case we can not assume that points are in general position, as for example in a (perfect) benzene ring at least 6 carbon atoms lie on a sphere. However, we have found that - probably due to both imprecision of data in the PDB and minor perturbations in atomic positions - all regions are tetrahedra.

3.2.1 Van der Waals and covalent edges in Delaunay tetrahedra

Here we characterize the edges of Delaunay tetrahedra into three classes:

(i) Edges, longer than the van der Waals bond distance of the two atoms in the vertices;

(ii) Edges, shorter than or equal to the van der Waals bond distance of the two atoms in the vertices;

(iii) Edges, shorter than or equal to the covalent bond distance of the two atoms in the vertices.

Clearly, type (iii) edges are also type (ii) edges, but the reverse is not true. Heuristically, the Delaunay tetrahedra will contain lots of atom pairs in bond distance. In this section we will discuss the contents of Table 3.1.

Definition: The *bond graph* of a tetrahedron in a Delaunay decomposition of a protein is defined as the graph, with vertices that correspond to the vertices of the Delaunay tetrahedron, and the edges are the atom pairs closer than their van der Waals bond distance. (Edge type (ii).)

Similarly *covalent bond graph* of a tetrahedron in a Delaunay decomposition of a protein is defined as the graph, with vertices that correspond to the vertices of the Delaunay tetrahedron, and the edges are the atom pairs closer than their covalent bond distance. (Edge type (iii).)

The columns of Table 3.1 correspond to the Delaunay tetrahedra where the connected vertices are in van der Waals distance, while the non-connected ones are longer than the atom-specific van der Waals distance (i.e. bond graphs). The rows correspond to the same graphs in the same order, but there the edges

0000	0011	0112	0222	1111	1113	1122	1223	2222	2233	3333	Vdw+cov	deg
26727667	26326279	15556169	107469	5095105	274198	7372956	168872	14529	90253	18226	count	cov
26727667	500990	179623	484	28576	28564	28165	586	6980	224	1	27501860	0000
	25825289	184357	106985	126110	7645	72264	90918	89	13744	76	26427477	0011
		15192189	0	0	10332	72875	62529	6624	63363	0	15407912	0112
			0	0	0	0	0	0	0	0	0	0222
				4930419	0	23626	14839	836	12922	18149	5000791	1111
					227657	0	0	0	0	0	227657	1113
						7176026	0	0	0	0	7176026	1122
							0	0	0	0	0	1223
								0	0	0	0	2222
									0	0	0	2233
										0	0	3333

Table 3.1: Frequency of bond graphs. The columns correspond to the bond graphs while the rows correspond to the covalent bond graphs. The first row and the last column contains the ordered degree sequences vertices. The second row shows frequency or respective bond graphs and the 12^{th} column shows the same for covalent bond graphs. The item in the intersection of a row and in a column contains the number of tetrahedra satisfying both of its row and its column.

correspond to vertices in covalent distance, and the non-edges vertex-pairs in non-covalent distance (i.e. covalent bond graphs).

The very first row and the last column describe the degrees of the vertices in the 4-vertex graphs in ascending order. Note that this is an invariant of the graph isomorphism class, and in case of 4 vertex graphs it is different for non isomorphic graphs. In what follows we will use this invariant to refer to the different isomorphism classes.

The second row shows frequency or respective bond graphs and the 12^{th} column shows the same for covalent bond graphs. The item in the intersection of a row and in a column contains the number of tetrahedra satisfying the definition both of its row and its column. Since van der Waals distances are larger than covalent distances, the lower left half of the table is empty. For example, on Table 3.1, in the intersection of column 1111 and row 0011, the number 126,110 means that from the more than 5 million tetrahedra with two van der Waals edges with degree-sequence 1111, only 126,110 contain exactly one covalent edge.

It is worth to mention, that while there are - albeit very few - length-3 and length-4 cycles in the bond graphs, there is not a single one in the covalent bond graphs (see rows 0222, 1113, 1122, 1223, 2222, 2233 and 3333 in Table 3.1), this observation correlates well with facts from basic biochemistry.

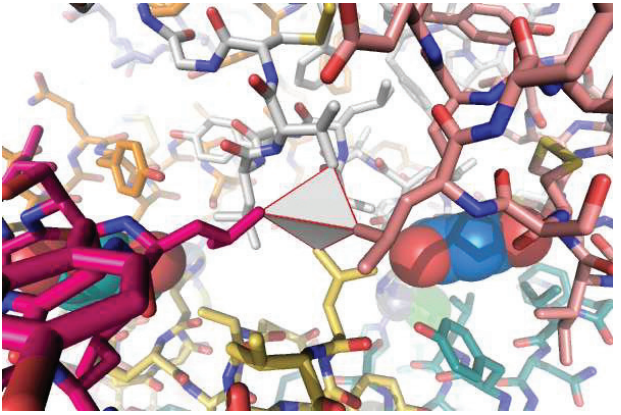


Figure 3.5: The only complete bond graph that has no covalent edges. (PDB entry: 1QIZ.)

Furthermore rows with degree-sequences 1113 and 1122 show that tetrahedra with at least 3 covalent bond edges do not admit further van der Waals edges. Similarly row with degree sequence 0112 shows that its intersection with column 0222 is 0, that is, a length-2 path of covalent edges prohibits a third van der Waals edge closing the path to a triangle.

On the other hand, the column 4444 on Table 3.1 shows that almost all complete 4-vertex van der Waals graph have two non-adjacent covalent edges. There is only a single complete 4-vertex van der Waals graph without covalent bonds.

This single exception can be found in an interesting configuration in the PDB entry 1qiz, in a human insulin hexamer structure. As we depicted on Figure 3.5, the four vertices of the tetrahedron consist of four carbon atoms, each in different polypeptide chains. More exactly, from the $C_{\delta 2}$ atom of the LEU^{13} of chain E, the $C_{\delta 2}$ atom of the LEU^{13} of chain K, the $C_{\delta 2}$ of the LEU^{17} of chain L, and $C_{\gamma 2}$ atom of the VAL^{18} of chain F.

3.2.2 Volume and tetrahedrality

In our test we - instead of examining the distribution of volume and tetrahedrality of regions separately - created density maps in both variables at the same time. The triple logarithmic plot can be seen on Figure 3.6. It is quite

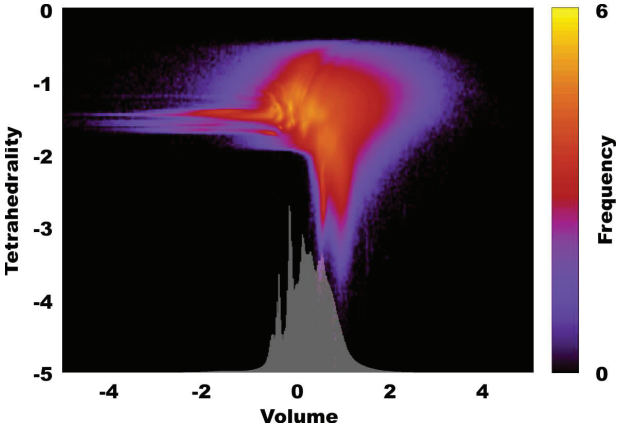


Figure 3.6: A point with coordinates (x, y) on the plot corresponds to all Delaunay regions whose volume is $10^{(x\pm0.01)}$ and tetrahedrality is $10^{(y\pm0.01)}$ and the color of the point corresponds to $\log(z+1)$ where z is the number of such regions. The white barplot on the bottom of the image is the same for volume only.

straightforward to see that at the boundary of the protein the tetrahedra tend to be more irregular and of larger volume, while in the inside of the protein, the tetrahedra are small, compact, and regular. (See Figure 3.2) However, the more intricate analysis depicted on Figure 3.6 shows a distinctly characteristic distribution. The fine structures in this plot indicate that a suitable classification of the simplices may decompose it into a sum of simpler plots. Although a complete characterization of the different peaks of the plot has not yet been found, in what follows we will show that certain properties of the tetrahedra separate some of them.

Labeling the vertices of the tetrahedra

First we examined tetrahedra grouped according to the set of atoms in their vertices. All tetrahedra were assigned a *vertex set* label that is the merging of the 4 symbols assigned with the elements in the corners in alphabetic order. (For example a tetrahedra spanned by a nitrogen, two carbon atoms and an oxygen would be assigned the symbol: 'C C N O'. Grouped by these labels, we listed the count of the tetrahedra in Table 3.2. Tetrahedron 'C C N O' (containing the peptide bound of amino acids) turns out to be the most frequent

Pattern	Count	Pattern	Count	Pattern	Count
C C N O	19,463,268	C C C O	13,979,006	C C C N	9,228,670
C C C C	8,549,030	C C O O	8,302,189	C N O O	7,148,317
C N N O	4,811,063	C C N N	4,137,294	C O O O	1,774,801
N N O O	983,656	N O O O	696,899	C C C S	575,423
C C O S	453,511	C N N N	320,021	C C N S	305,453
C N O S	255,407	O O O O	220,453	N N N O	184,983
C O O S	99,173	C C S S	56,480	C N N S	42,572
N O O S	30,644	C O S S	23,276	N N N N	21,076
C N S S	19,843	N N O S	16,119	O O O S	8,380
C C C SE	7,624	N O S S	4,995	C C O SE	4,582
C C N SE	2,822	C N O SE	2,289	N N N S	1,982
N N S S	1,872	C S S S	1,848	O O S S	1,565
N S S S	793	C O O SE	764	C N N SE	433
S S S S	420	O S S S	335	C C C F	256
N O O SE	230	C C F O	224	N N O SE	149
C O O P	145	C C F N	123	O O O P	101
C C SE SE	99	C F N O	96	N O O P	91
C C S SE	72	O O O SE	70	C C C I	65
C C I O	51	C F O O	47	C CL N O	40
C N O P	38	N N N SE	31	C I O O	28
C C CL N	27	C C CLO	26	C O S SE	25
AS C C S	21	AS C C O	20	C I N O	20
C N SE SE	19	AS C C C	17	C F N N	16
C C O P	15	AS C O S	15	C C C CL	15
F N O O	14	C O O V	12	C C I N	12
AS C N O	11	B C O O	10	C CLO O	10
AS C C N	10	C O SE SE	9	C C F S	9
O O O V	8	F N N O	6	C C I S	6
N N O P	6	AS C O O	6	AS N O O	5
C N S SE	5	B C N O	4	N N SE SE	4
B C C O	4	CL N O O	4	I O O O	4
I N O O	4	CL N N O	3	N O SE SE	3
F O O O	3	AS N N O	3	AS C N N	3
N O S SE	3	C I O S	2	C C F F	2
B N O O	2	C O P S	2	C F O S	2
AS C N S	1	O O S SE	1	C F F N	1
C C C P	1	O O P S	1	N N S SE	1
AS O O S	1	AS N O S	1	C CL N N	1

Table 3.2: The counts of different types of Delaunay tetrahedra in the test set of 5,757 PDB entries.

with 19,463,268 occurrences in our test set. The frequency of other labels decrease exponentially.

We observed that splitting the distribution plot according to the composition of the vertex-sets of the Delaunay tetrahedra would show different patterns for different labels. These plots are listed in Appendix 1 for vertex sets appearing at least 500 times in our databasis. Plots for less frequent vertex sets seem to be inconclusive scattered data. As an example for large differences between shape/volume distributions compare 'C C N O', 'C C O S'. 'C N O S' and 'N N O O'.

Volume and tetrahedrality distributions split bond graphs

Similarly we can split the distribution plot in Figure 3.6 with the (covalent) bond graph classification introduced in section 3.2.1. The results are more

pronounced than in the case of corner labels.

We have showed in section 3.2.1, that only 6 of the possible graphs on 4 vertices appear as covalent bond graphs. In Appendix 2 the corresponding distribution plots are included. In contrast all of the possible 4 vertex graphs appear as (general) bond graphs, and respective plots are shown in Appendix 3.

3.2.3 Ligand atoms in tetrahedra

Here we analyze the atomic environments of ligand atoms, bound to proteins. The atomic environment of each ligand atom will be identified as the vertices of a tetrahedron in a tetrahedral decomposition of the heavy atoms of the protein, containing the atom of the bound ligand.

By this approach we can describe uniformly and in a discreet manner the environment of ligand atoms in proteins. The classification is given by describing tetrahedra according to the atoms in their vertices, and by the atoms of the ligands the convex hull these tetrahedra contain (on Figure 3.2 the ligand is shown in purple).

We are using the ligand-identification technique described in [27], using the classification of monomer Id's given in [22] and [21]. Concisely, we doubly checked if a ligand, even with more than one monomer Id's is one molecule or not, by comparing the bond tables from mmCIF and the atomic distances. The ligand was thrown out if recognized as a crystallization artifact, covalently bound (but non-protein-) or junk molecule [22].

In Table 3.3 we present the frequency of different types of tetrahedra containing metallic ligand atoms, with the notation introduced in section 3.2.2. Tetrahedra are being classified based on the atom types appearing in their vertices. Table 3.4 is similar, but with the frequent non metallic ligand atoms.

	C C N O	C N O O	C N N O	C O O O	C C N N
Zn	5	7	3	2	0
Mg	1	11	4	1	0
Ca	0	0	0	0	0
Mn	0	0	0	3	1
Fe	2	3	0	1	1

	N N O O	N O O O	C N N N	N N N O	O O O O
Zn	14	14	0	38	2
Mg	15	10	6	5	6
Ca	0	1	0	2	48
Mn	3	4	0	4	5
Fe	0	0	2	0	0

	C C C S	N N N N	C C N S	C C S S	N O S S
Zn	0	0	4	0	32
Mg	0	5	0	0	0
Ca	0	0	0	0	0
Mn	0	0	0	0	0
Fe	2	1	0	5	0

Table 3.3: The classifications of the tetrahedra around metal ligand atoms. The tetrahedra not present contain no metal atoms.

	C C N O	C C C O	C C O O	C N O O	C C C C	C N N O
H	5590	5385	5461	4678	3091	2651
C	4218	4289	3757	3295	2628	1806
O	1673	823	1097	1470	345	1373
N	585	554	589	605	195	220
P	41	10	17	30	6	110
S	77	42	43	49	27	31
F	27	40	42	22	31	14

	C O O O	C C C N	C C N N	N N O O	N O O O	C N N N
H	2899	2360	1304	1328	1334	663
C	1777	2091	1125	839	886	422
O	621	601	623	731	519	524
N	447	307	97	150	187	36
P	17	18	38	64	28	70
S	16	28	21	9	9	6
F	6	18	5	5	2	0

	N N N O	O O O O	C C O S	C C C S	N N N N	C N O S
H	583	665	325	298	139	204
C	317	276	226	267	70	132
O	521	133	47	41	214	92
N	40	170	56	29	6	39
P	75	4	1	0	69	0
S	5	3	9	5	1	4
F	2	0	0	2	0	1

	C O O S	C C N S	N N O S	N O O S	C N N S	C C S S
H	187	149	88	66	32	16
C	107	133	50	32	30	59
O	37	45	31	20	29	8
N	33	19	7	7	4	0
P	0	0	0	1	2	0
S	4	1	1	0	0	0
F	0	0	0	0	0	0

	N O S S	O O O S	C N S S	N N N S	O O S S	N N S S
H	2	18	19	5	7	9
C	3	11	11	9	7	2
O	8	3	1	7	0	4
N	2	2	0	0	7	0
P	0	0	0	0	0	0
S	0	0	0	0	0	0
F	0	0	0	0	0	0

	C O S S	C N O SE	C N N SE
H	7	1	1
C	2	0	0
O	2	0	0
N	3	0	0
P	0	0	0
S	0	2	2
F	0	0	0

Table 3.4: The classifications of the tetrahedra around metal ligand atoms. The tetrahedra not present contain no metal atoms.

Chapter 4

Database cleaning methods

The fast growing Protein Data Bank (PDB) contains a vast amount of 3-dimensional experimental information on protein- and nucleic-acid structures. In 1971 - when Meyer and Hamilton founded PDB - the primary objective was to provide a depository of structures for scientists investigating a small number of proteins at a time. Today the Protein Data Bank contains 50,000 entries, and it is a rich repository of the structural biological information, without which our work presented in the previous chapters would have been impossible.

Unfortunately though, every scientist working with PDB comes across lots of inconsistencies in the data. The inconsistencies have various sources, from simple human errors, oversight, misinterpretation of PDB file format, to deliberately erroneous syntax to overcome certain limitations of the PDB depositing rules. The automated processing of a larger subset of entries is largely hindered by these shortcomings of the format.

As a large number of these errors are easy to detect even with simple Perl scripts, there has been a number of attempts to correct these errors with computers. Unfortunately every such approach has its shortcomings, and usually ends up in converting easy to detect syntax errors into hard to detect semantic errors. As most bio-informatics research groups need to run data mining algorithms on the PDB, discovering and correcting errors is very important. Mostly data cleaning is done in house by each research group, errors and cleaning methods are rarely get reported to the public.

The most notable attempts towards correcting PDB entries are the in house corrections of the RCSB PDB [2], the transition to the mmCIF format, and PROCHECK [17]. Most of the corrections by RCSB PDB are a response to

user reported errors. Since entries are edited by a limited number of employees, it takes time for each reported error to appear in the official PDB. The mmCIF format was introduced to overcome the limitations of the PDB file format by Hall, Allen and Brown [26]. During conversion the PDB files are parsed and written into syntax correct mmCIF files, and can be interpreted as a syntax correction of the PDB.

PROCHECK and PROCHECK-NMR evaluates the quality of a protein structure. They generate plots for analyzing the residue-by-residue geometry of protein structures deposited in the PDB. Note however, that the results produced by PROCHECK only suggest how usual (or unusual) the protein structure is, as compared with other high resolution structures. When PROCHECK highlights an unusual region, it may be an error, or an interesting feature.

As an example for inhouse corrections, let us mention our own enhanced PDB version created by Zoltan Szabadka, the RS-PDB. Although the methods used were briefly presented as a conference poster, the RS-PDB is available exclusively by members of the PIT Group. The RS-PDB converts the object oriented mmCIF format into a relational SQL database using the star scheme. During the conversion all entries were parsed, interpreted and rigorously checked for errors with our conversion software.

WikiPDB (`http://wikipdb.org/`) is taking a different approach: we provide a Wikipedia-like interface (an extension of the Mediawiki software) for modification of the data bank files, and discussion between scientists doing research on the same structures. Our system detects syntax errors, provides an API for semantic error checking plug-ins, and keeps track of different versions.

The most easy to discover type of error in PDB files, are syntax errors. PDB files have an extremely strict syntax which is rather unforgiving for errors, and that concludes in uncomfortable limitations. Unlike modern object based data storing models like XML, the PDB format uses fixed length strings, and the fields within a row are defined by character position within the line. That can result in two types of error: data overflow and data dislocation. For example PDB has been converted into syntax correct mmCIF format files, but some of the syntax errors were converted into semantic errors.

Other type of errors can result from mistyped characters, improper interpretation of the experimental data, or simply the misunderstanding of PDB file format specifications. These errors are usually easy to discover automatically, but may prove hard - or even impossible - to correct.

During automated processing of PDB files, one bumps into many of these errors, while trying to interpret the data. Correcting errors in hundreds of PDBs at once manually is infeasible for one research group, and automatic error correction can end up making things worse, by changing an error we have been able to find into an error that is not to be recognized for a long time. In situations like that it would be useful if one could discuss those errors with experts of the individual PDBs at once. On the other hand experts involved with a particular PDB may find it beneficial, if they would be notified about those errors, and could share their opinions about possible corrections.

4.1 WikiPDB

WikiPDB is a Wiki based community website intended for scientists involved in working with PDB files. Our aim is to bring together two types of research technique: automated mass processing, and manual examination of single PDBs. Those processing large number of PDBs with algorithmic methods can call attention to errors humans would not find normally, while researchers involved in handling single PDBs can discuss and correct errors.

Our implementation is an extension of the media wiki software. Each PDB entry has a separate page in the pdb name space, and an extra tab containing the data history. Each version can be downloaded, changed, checked for errors, and anyone can upload new revisions.

It's very important to note, that Wiki based data storage is vulnerable to malicious activity. For that reason WikiPDB does not open a specific revision by default, but encourages the user to choose a revision based on the change log, and verify it before using. The main purpose of WikiPDB is to encourage communication between researchers, and help correcting errors, the need for checking the validity of data is the price to pay for that. When referencing a modified entry from the WikiPDB, reference the original PDB, and the changes applied to it as well.

PDB files are regulary syncronized with RCSB PDB by "Importbot": a weekly ran script on the WikiPDB server. The latest version created by Importbot is always identical to the one in RCSB PDB. In the future we plan to develop a feedback system, that would email the list of errors that were confirmed by scientists working with the structure, along with a proposed correction. Ideally the proposed correction should be based on a consensus

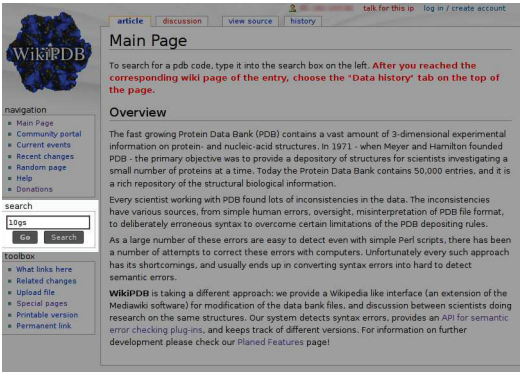


Figure 4.1: Screenshot of the main page of WikiPDB, with the search box highlighted

between experts of the specific structure. This way the result of laborious data cleaning would find it's way back to the official - and due to its non Wiki nature, more trustworthy - RCSB PDB.

To search for a PDB visit `http://wikipdb.org/`, and after typing the PDB code into the search box press the "GO" button. WikiPDB will load a page containing human readable information on the PDB file. (In most cases the abstract of the related article.). Go to the "Data history" tab, choose a suitable revision, and press modify to start editing it. Once ready with changes press "Error check data" to submit the new revision for error checking, or "Submit data" to save changes into a new revision. Note that data upload may take a few minutes.

4.1.1 WikiPDB API

WikiPDB has an open API for error check plug-ins, that allows third party research groups to implement error checking algorithms that work together with WikiPDB. As plug-ins can be hosted by their developers the actual implementation of the algorithm can be kept as a secret of the developer.

An error check plug-in is a simple application (preferably written in php) that is capable of communicating with the WikiPDB through the API described below, and discovers errors (or suspicious records that might be errors, but may as well be correct) in PDB files.

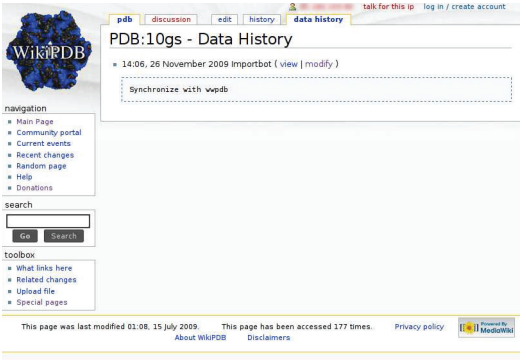


Figure 4.2: Screenshot of a data history tab in WikiPDB. Notice that the only version so far for this particular PDB is the one imported from RCSB PDB.

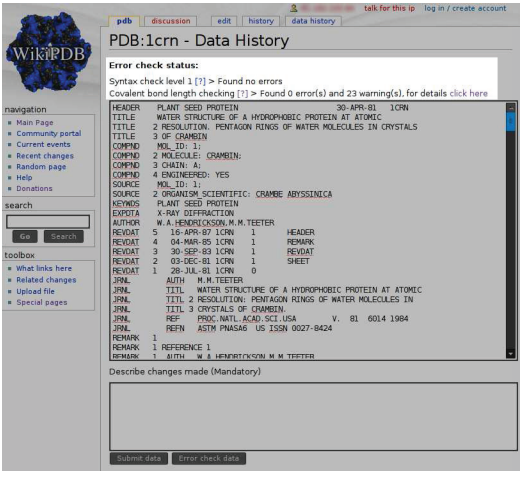


Figure 4.3: Screenshot of an edit page in WikiPDB after an error checking has finished. The result of the error checking is highlighted. In this case there were no syntax errors, but 23 atom-atom distances were reported to be slightly longer or shorter than expected.

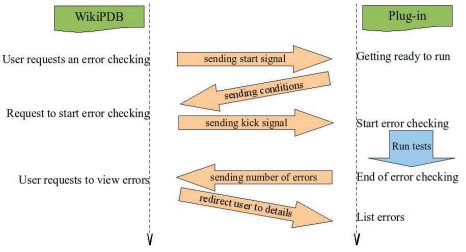


Figure 4.4: The communication between WikiPDB and a plug-in.

To make writing plug-ins easy, we provide a simple PHP abstract class, which everyone is welcome to use without any restrictions. All the functions necessary for creating a plug-in have been implemented into this class; what remains for the developer to do is write the function responsible for detecting errors, report these errors, and write the documentation for the plug-in. The verifying of input variables has also been implemented into the constructor of this class, along with the processing start and kick signals. The former signal is responsible for starting the plug-in, while the latter initiates the verification process. To make processing of the information easier, the lines of the PDB file are also available as an array. Finally, setting up a relation of interdependency between the plug-ins is possible, so that whether a plug-in is initiated at all might be made contingent on the result of another plug-in.

Communication protocol

As both browsers and HTTP servers may limit the size of post variables it was necessary to use a sophisticated communication protocol. Both for saving and errorchecking the PDB file has to be uploaded on the webserver first. For that we use an AJAX based method: after sending the number of rows, and the command (save or error check) the content of the message box is being split up to 300 row long chunks, and sent separately. This technique also allows us to monitor upload progress. After sending is complete the server will start processing the data.

In case of error checking the next step is to "load" the error check plug-ins. The server sends a "start" signal to each of the registered plug-ins, and expects a list of dependencies. After all plug-ins answered (or failed to answer

```
<?
require_once("./inc/AbstractErrorCheckClient.php");
require_once ("./inc/httpClient.php" );
class TrivialErrorCheckClient extends AbstractErrorCheckClient
{
    protected function userDefinedErrorcheck()
    {
        $retVal['errors']=0;
        $retVal['warning']=0;
        return $retVal;
    }
    protected function help()
    {
        echo "";
    }
    protected function details()
    {
        echo "";
    }
    protected function apiDetails()
    {
        echo json_encode(array());
    }
    protected function useraction()
    {

    }
}
try
{
    new TrivialErrorCheckClient("","");
}
catch(Exception $e)
{
    echo "Exception > ".$e->getMessage();
    exit();
}
?>
```

Figure 4.5: Minimal sourcecode for a plug-in based on the AbstractErrorCheck-Client class.

within time limit) the server sends a kick signal to the plug-ins having no dependencies. The plug-ins can download the PDB file to be checked, and start the work. Every time a plug-in is ready, it sends the number of errors and warnings to the server, and the server sends the kick signal to the plug-ins, whose dependencies got satisfied at this point.

In the meantime, the browser client periodically invokes an AJAX request that retrieves results from the server. This way the user is informed about the progress made by the plug-ins, and it is possible to start evaluating the results given by the plug-ins that finished working. The list of errors is linked from the browser, and it is retrieved directly from the plug-in, without intervention from the WikiPDB server.

Writing a PHP plug-in based on the AbstractErrorCheckClient class

Figure 4.5 shows the minimal sourcecode for a plug-in based on the AbstractErrorCheckClient class. The easiest way of writing a WikiPDB plug-in, is to extend this code as described below.

The most important function is the userDefinedErrorcheck. This is where

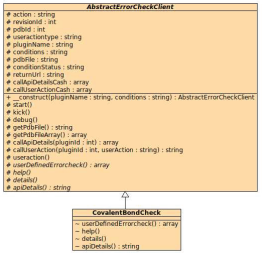


Figure 4.6: The UML diagram of the abstract class and the covalent bond checking plug-in.

the actual error checking takes place. The return value should be an associative array that has a numeric value for the 'errors' and 'warning' keys. To download the PDB file simply use "$pdbfile=$this->getPdbFileArray();", which will import the rows of the PDB file into rows of the $pdbfile array. Results of the error checking algorithm should be stored at the end.

The second function - "help()" - should return a human readable HTML file containing a description of the plug-in to the browser client. The third and forth - "details();" and "apiDetails()" - should return a description of each error found in the file. The only difference is that "details()" should return human readable informations in HTML format, while "apiDetails()" should return a JSON encoded array.

The function "useraction();" is optional. User actions are intended to provide an interface for communication between plug-ins. Another way to utilize user actions, is to provide additional information pages for WikiPDB users apart from the details page.

Finally the constructor of the class expects two arguments: the name of the plug-in, and a list of dependencies. The list of dependencies should be given as a comma separated list of plug-in ids.

A detailed tutorial on writing two simple plug-ins - along with the files that need to be included - is available at http://wikipdb.org/index.php/WikiPDB:Api

4.1.2 Bond length checking

It is very simple to discover syntax errors, so our "proof of concept" plug-in for WikiPDB was a syntax error checking plug-in. As a next step we decided

to provide a semantic error checking plug-in. Despite of the fact, that atom-atom distance is strongly related to covalent bonding, in our earlier research [27] we discovered covalent bond length errors in PDB files, hence bond length checking proves to be an effective way to filter incorrect structures.

PDB files are carriers of data acquired through X-ray diffraction or NMR measurement, concerning the spatial coordinates of the atoms of protein and nucleic-acid structures. The accuracy of such information depends on the resolution of the measurement procedure. Nevertheless, measurement errors as well as defective connectivity records might occur. Such error might be a case of ill-ascertained coordinates, for example. The resulting flaws in structure might be filtered out by checking if the length of bonds between the atoms is appropriate. The plug-in presented in this section checks covalent bonds. It cross-examines the bond-length calculated from the measured spatial structure coordinates with its own estimateion of bond length, which is based on the covalent radius of the atoms. Our procedure takes advantage of the fact that the sum of the covalent radii of two atoms should, allowing for a certain margin of error, be equal the length of covalent bonds between them.[1] Should the difference between the length of bond calculated from the coordinates and the length of bond prescribed by the amino-acid structure be more than 5%, the user receives a warning. If the difference is higher than 10%, the program reports an error. The plug-in also takes into consideration the covalent radius of the atom, and examines if the covalent radii of the two atoms intersect. If so, it also checks whether this has been registered in the connectivity record.

The first step of the algorithm builds up the graph of covalent bonds within the amino acids contained in the molecule. This is performed according to a chart, which contains corresponding information regarding the atoms participating in the covalent bond to all twenty kinds of amino-acid.

The second step is to check if atom-atom distances in the whole molecule are consistent with this graph and the connectivity records. The naive algorithm would be to go through all $O(n^2)$ atom pairs, calculate their distances, and compare it with binding or non binding distances respectively. Unfortunately it is not possible to decrease worst case time complexity, as the output may be as big as $O(n^2)$. (All atoms set to $(0, 0, 0)$ will do as an example of that.) But we can give an output sensitive algorithm, that will have a time complexity of $O(n \log n + e)$ where e is the number of reported errors. What's more, in case we are interested in only the first k errors (k is an arbitrary constant), then

the time complexity becomes $O(n \log n)$.

The covalent binding graph may not have more than $O(n)$ edges, as each atom can only take part in a low number of covalent reactions. (In a typical PDB the constant can be chosen to be 4, as carbons have the most covalent bindings from the atoms that regularly appear in proteins and their ligands.) Hence checking if all of the binding atom pairs are in correct distances takes up $O(n)$ time, if we loop through all edges.

Now we have to check if non bonding atoms are far enough from each other. A natural idea would be to use kd-trees for that procedure, but Lee [18] states that a query for an axis parallel range has a time complexity of $O(n^{\frac{2}{3}} + p)$ (p denotes the number of points reported) which would result in a time complexity $O(n^{\frac{5}{3}} + e)$ for the entire algorithm.

Let ε denote the minimal length of a valid covalent bondage and ρ be the maximal length of any covalent bond. Let $M^{k,\rho}_{l,m,n}$ denote the range $[(l-k)\rho, (l+k+1)\rho] \times [(m-k)\rho, (m+k+1)\rho] \times [(n-k)\rho, (n+k+1)\rho]$ where l, m, n are integers. We will store those $M^{0,\rho}_{l,m,n}$ ranges in a balanced binary tree that contains an atom. While building the tree, store the list of atoms contained in a range. Time complexity for that is $O(nlogn)$.

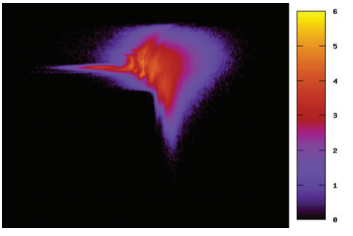
Now check each atom pair where both atoms are taken from an $M^{1,\rho}_{l,m,n}$ range. (It is enough if the first is taken form the smaller $M^{0,\rho}_{l,m,n}$) As theoretically non-binding atom pairs participating in a bond error have to be in the same, or neighboring $M^{0,\rho}_{l,m,n}$ we will find all errors.

The time complexity of listing atoms in $M^{1,\rho}_{l,m,n}$ is $O(\log n + n_{l,m,n})$ where $n_{l,m,n}$ denotes the number of atoms in the range. The time complexity of checking each pair of atoms in the rage is $O(n^2_{l,m,n})$. If we now split $M^{1,\rho}_{l,m,n}$ into ε sized ranges, then one of these ranges will contain $cn_{l,m,n}$ points accounting for at least $cn^2_{l,m,n}$ bond errors, where $c = (\varepsilon/\rho)^3$ is a constant. Hence the algorithm will not exceed the time complexity $O(n \log n + e)$.

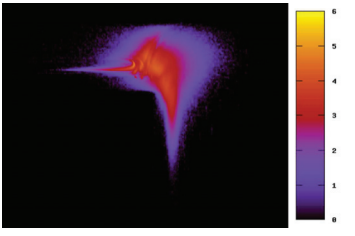
Chapter 5

Appendix 1

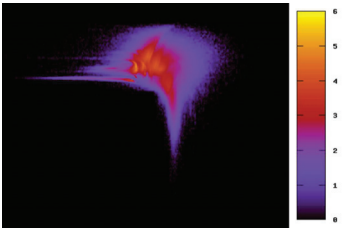
This appendix shows the empirical distribution plots of the volume and tetrahedrality of certain types of tetrahedra as explained on page 46 in section 3.2.2. Tetrahedrality and volume intervals are the same as on 3.6. The plots are ordered by the frequency of the vertex set, and only those are shown where frequency was more than 500.



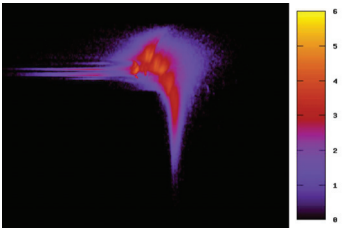
Vertex set: C C N O, Frequency: 19463268



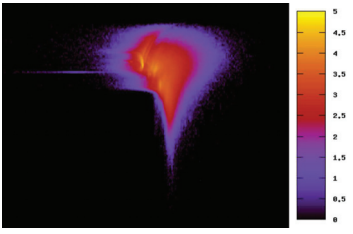
Vertex set: C C C O, Frequency: 13979006



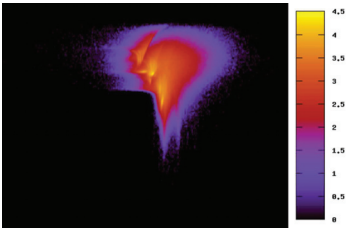
Vertex set: C C C N, Frequency: 9228670



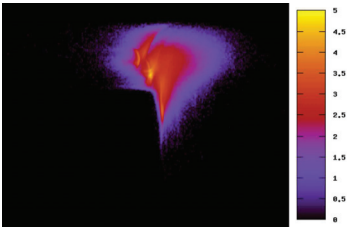
Vertex set: C C C C, Frequency: 8549030



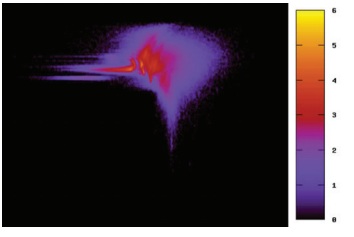
Vertex set: C C O O, Frequency: 8302189



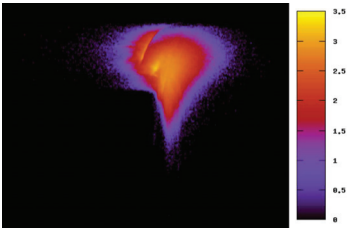
Vertex set: C N O O, Frequency: 7148317



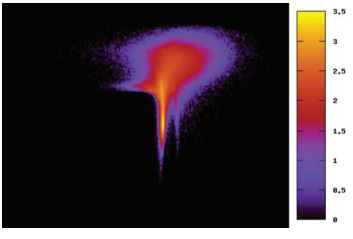
Vertex set: C N N O, Frequency: 4811063



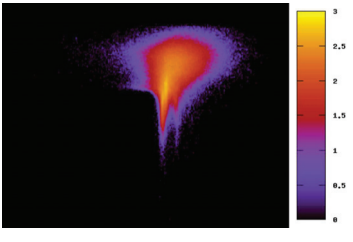
Vertex set: C C N N, Frequency: 4137294



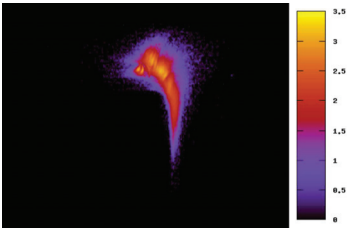
Vertex set: C O O O, Frequency: 1774801



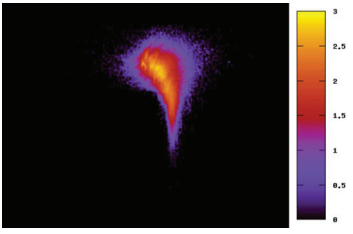
Vertex set: N N O O, Frequency: 983656



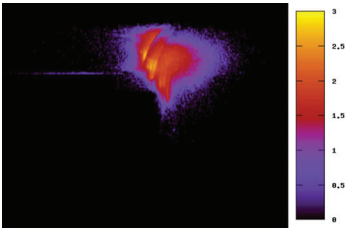
Vertex set: N O O O, Frequency: 696899



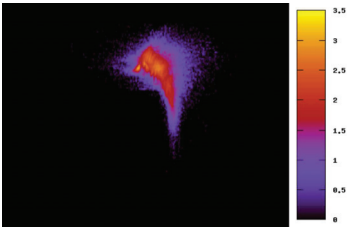
Vertex set: C C C S, Frequency: 575423



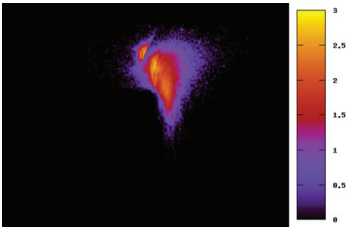
Vertex set: C C O S, Frequency: 453511



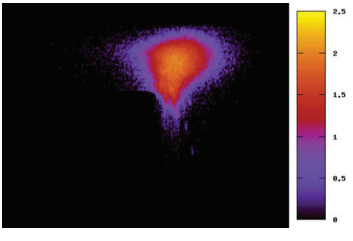
Vertex set: C N N N, Frequency: 320021



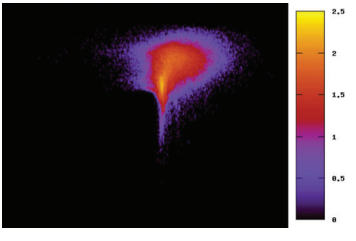
Vertex set: C C N S, Frequency: 305453



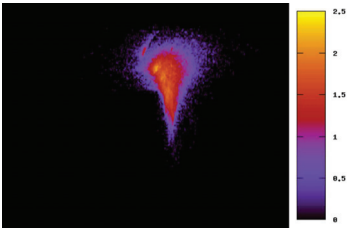
Vertex set: C N O S, Frequency: 255407



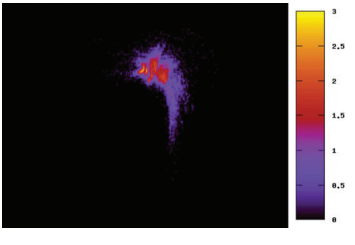
Vertex set: O O O O, Frequency: 220453



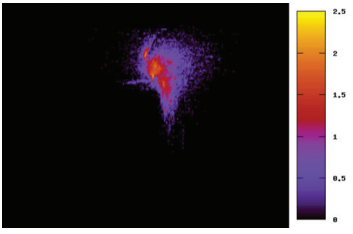
Vertex set: N N N O, Frequency: 184983



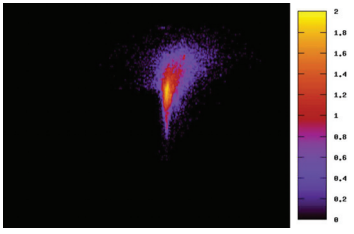
Vertex set: C O O S, Frequency: 99173



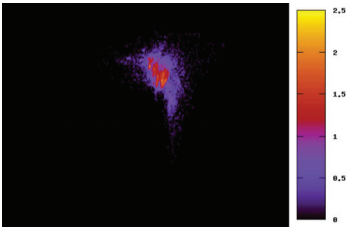
Vertex set: C C S S, Frequency: 56480



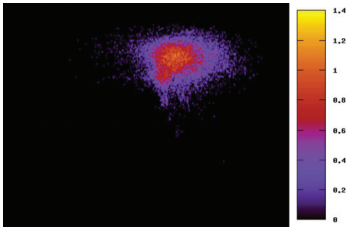
Vertex set: C N N S, Frequency: 42572



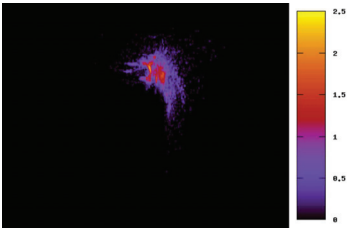
Vertex set: N O O S, Frequency: 30644



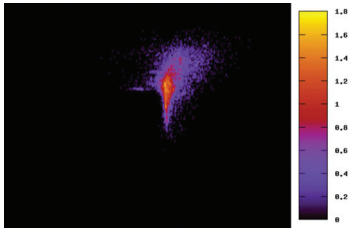
Vertex set: C O S S, Frequency: 23276



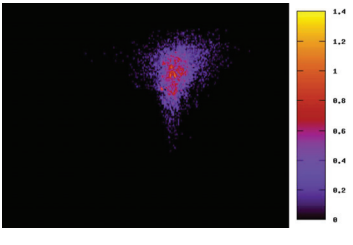
Vertex set: N N N N, Frequency: 21076



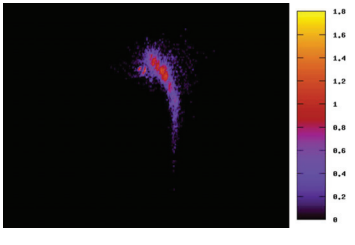
Vertex set: C N S S, Frequency: 19843



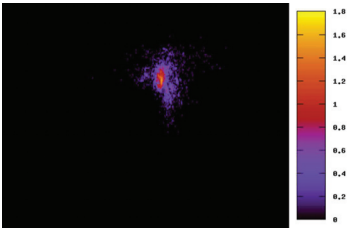
Vertex set: N N O S, Frequency: 16119



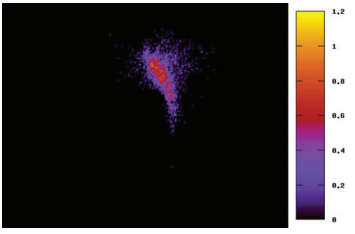
Vertex set: O O O S, Frequency: 8380



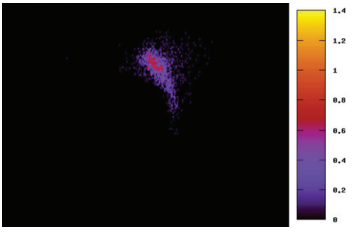
Vertex set: C C C SE, Frequency: 7624



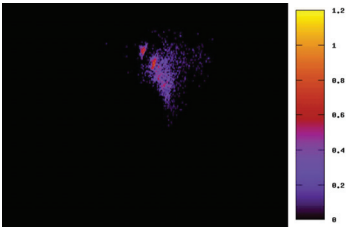
Vertex set: N O S S, Frequency: 4995



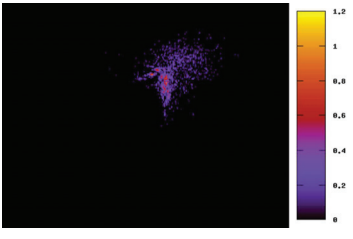
Vertex set: C C O SE, Frequency: 4582



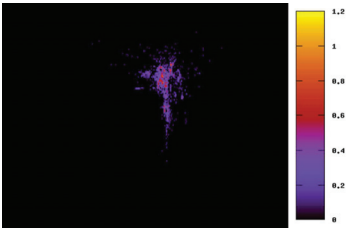
Vertex set: C C N SE, Frequency: 2822



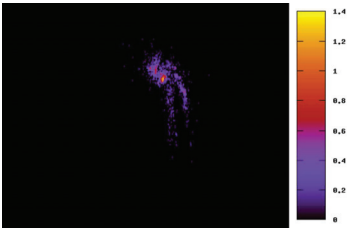
Vertex set: C N O SE, Frequency: 2289



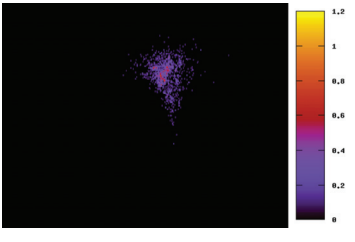
Vertex set: N N N S, Frequency: 1982



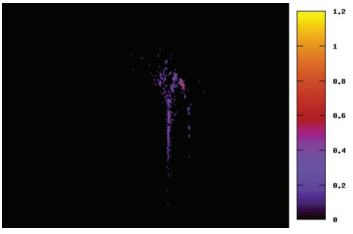
Vertex set: N N S S, Frequency: 1872



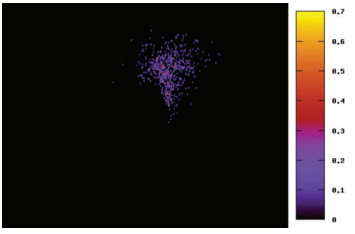
Vertex set: C S S S, Frequency: 1848



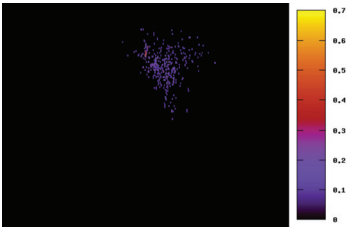
Vertex set: O O S S, Frequency: 1565



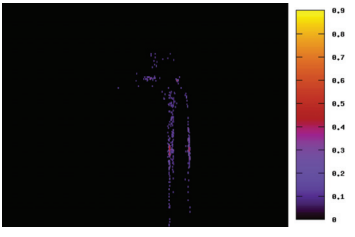
Vertex set: N S S S, Frequency: 793



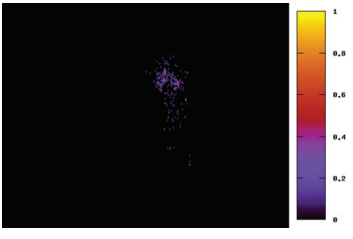
Vertex set: C O O SE, Frequency: 764



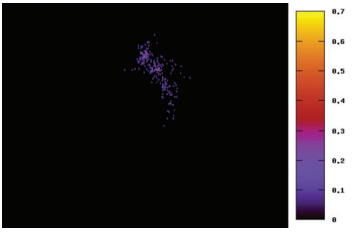
Vertex set: C N N SE, Frequency: 433



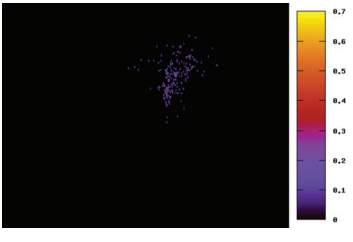
Vertex set: S S S S, Frequency: 420



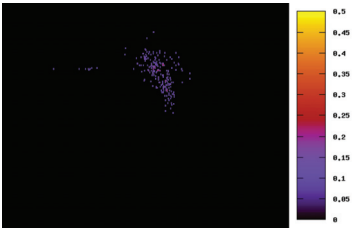
Vertex set: O S S S, Frequency: 335



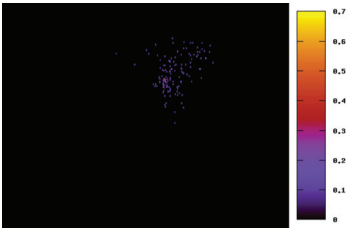
Vertex set: C C C F, Frequency: 256



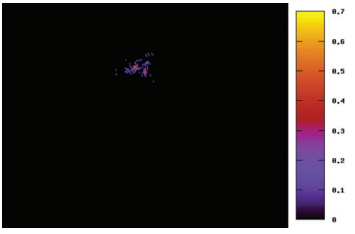
Vertex set: N O O SE, Frequency: 230



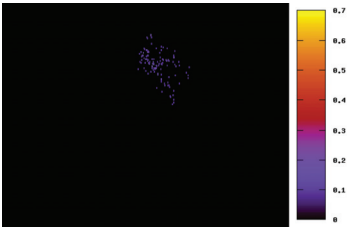
Vertex set: C C F O, Frequency: 224



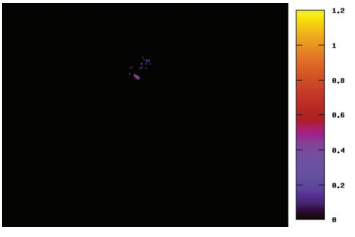
Vertex set: N N O SE, Frequency: 149



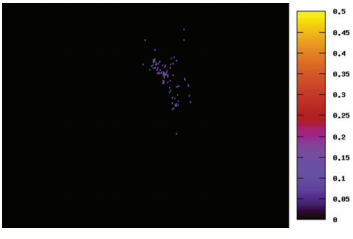
Vertex set: C O O P, Frequency: 145



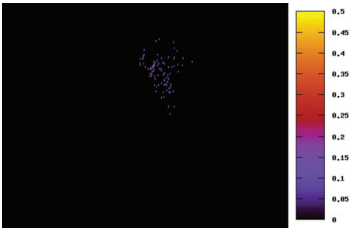
Vertex set: C C F N, Frequency: 123



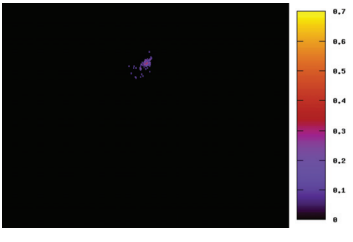
Vertex set: O O O P, Frequency: 101



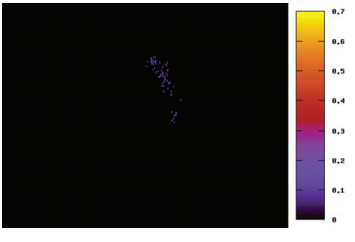
Vertex set: C C SE SE, Frequency: 99



Vertex set: C F N O, Frequency: 96

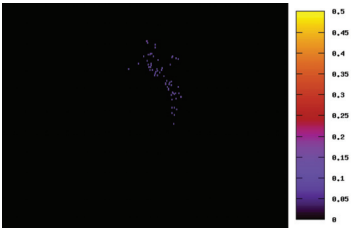


Vertex set: N O O P, Frequency: 91

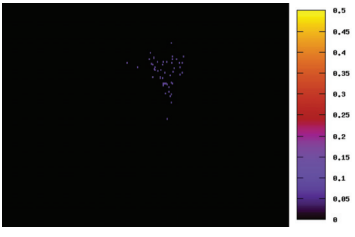


Vertex set: C C S SE, Frequency: 72

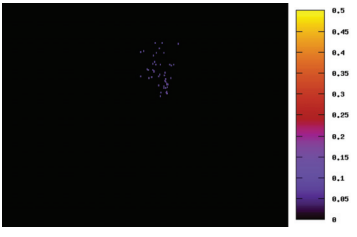
Vertex set: O O O SE, Frequency: 70



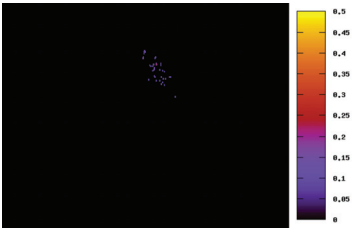
Vertex set: C C C I, Frequency: 65



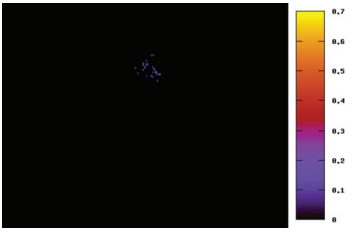
Vertex set: C C I O, Frequency: 51



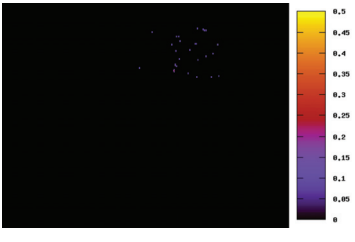
Vertex set: C F O O, Frequency: 47



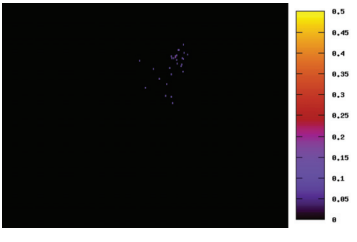
Vertex set: C CL N O, Frequency: 40



Vertex set: C N O P, Frequency: 38



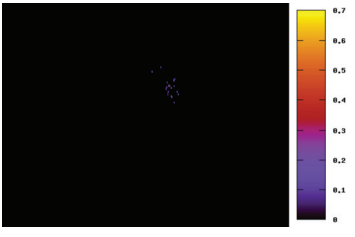
Vertex set: N N N SE, Frequency: 31



Vertex set: C I O O, Frequency: 28

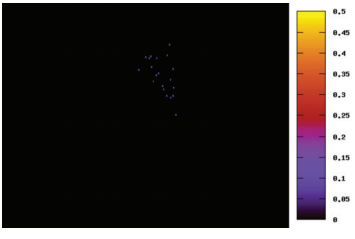
Vertex set: C C CL N, Frequency: 27

Vertex set: C C CL O, Frequency: 26

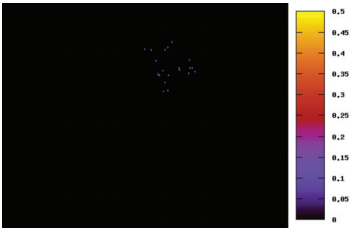


Vertex set: C O S SE, Frequency: 25

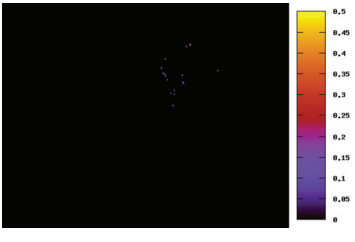
Vertex set: AS C C S, Frequency: 21



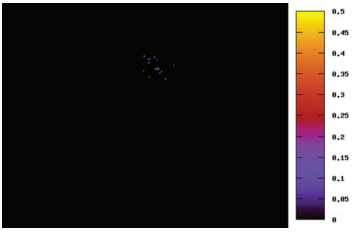
Vertex set: AS C C O, Frequency: 20



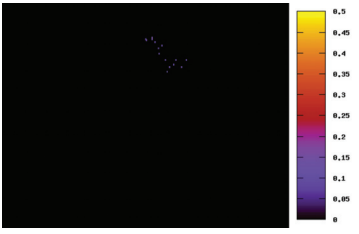
Vertex set: C I N O, Frequency: 20



Vertex set: C N SE SE, Frequency: 19

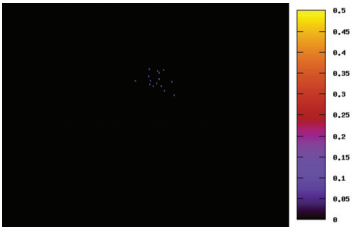


Vertex set: AS C C C, Frequency: 17

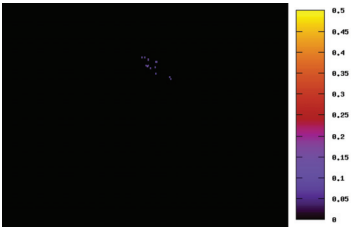


Vertex set: C F N N, Frequency: 16

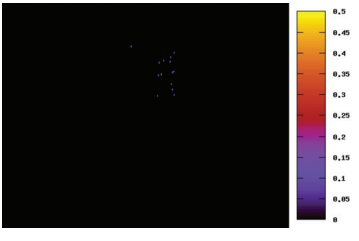
Vertex set: C C O P, Frequency: 15



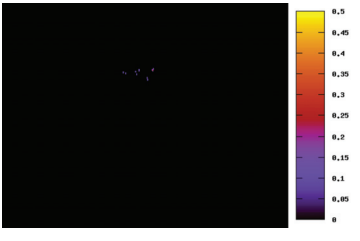
Vertex set: AS C O S, Frequency: 15



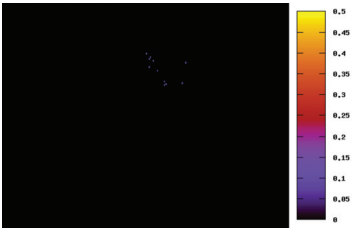
Vertex set: C C C CL, Frequency: 15



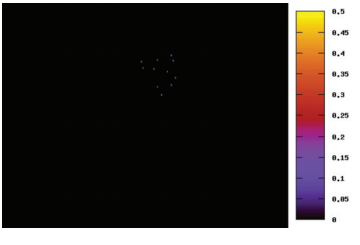
Vertex set: F N O O, Frequency: 14



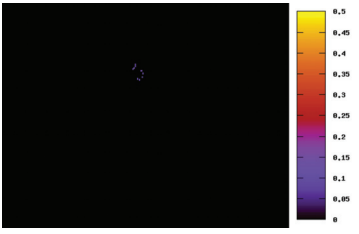
Vertex set: C O O V, Frequency: 12



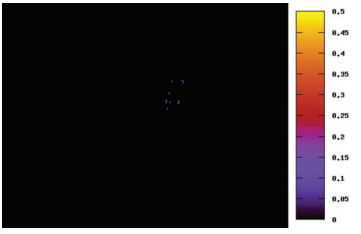
Vertex set: C C I N, Frequency: 12



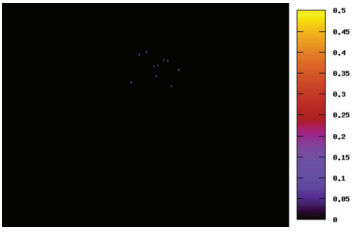
Vertex set: AS C N O, Frequency: 11



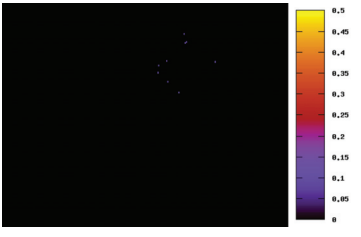
Vertex set: B C O O, Frequency: 10



Vertex set: C CL O O, Frequency: 10

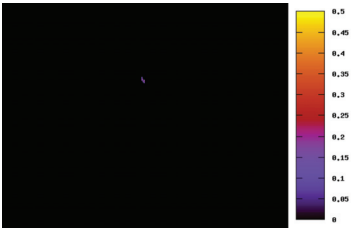


Vertex set: AS C C N, Frequency: 10

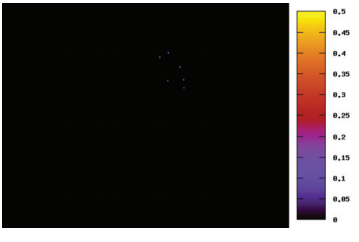


Vertex set: C O SE SE, Frequency: 9

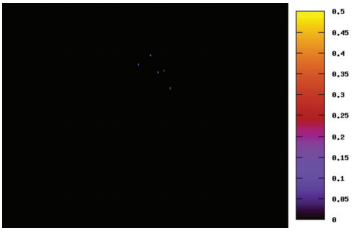
Vertex set: C C F S, Frequency: 9



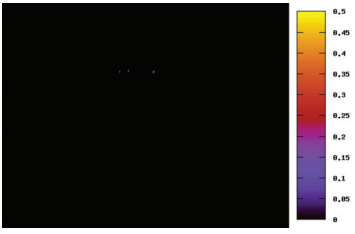
Vertex set: O O O V, Frequency: 8



Vertex set: F N N O, Frequency: 6



Vertex set: C C I S, Frequency: 6

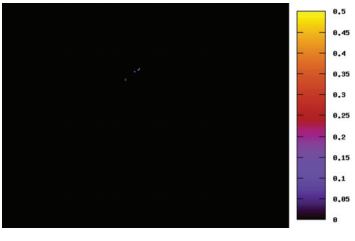


Vertex set: N N O P, Frequency: 6

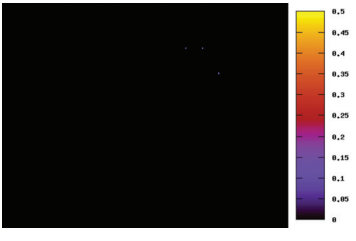
Vertex set: AS C O O, Frequency: 6

Vertex set: AS N O O, Frequency: 5

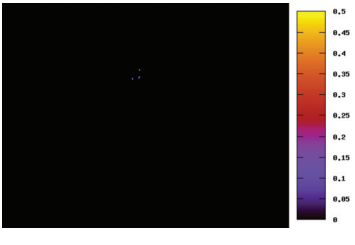
Vertex set: C N S SE, Frequency: 5



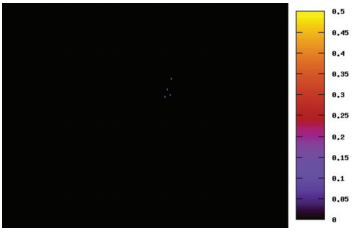
Vertex set: B C N O, Frequency: 4



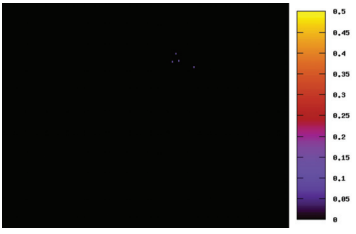
Vertex set: N N SE SE, Frequency: 4



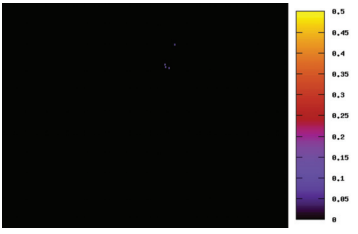
Vertex set: B C C O, Frequency: 4



Vertex set: CL N O O, Frequency: 4



Vertex set: I O O O, Frequency: 4

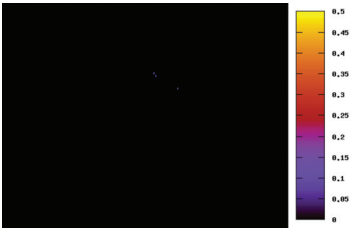


Vertex set: I N O O, Frequency: 4

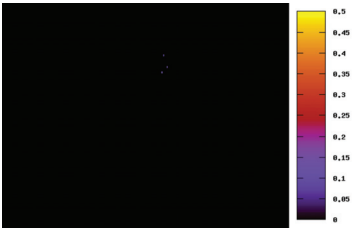
Vertex set: CL N N O, Frequency: 3

Vertex set: N O SE SE, Frequency: 3

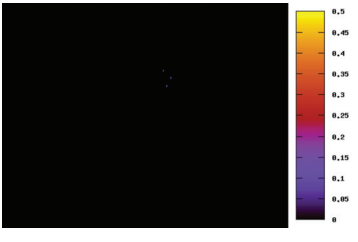
Vertex set: F O O O, Frequency: 3



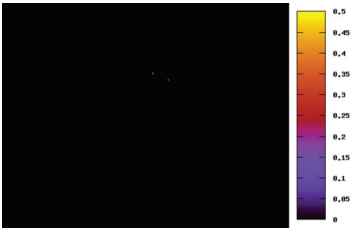
Vertex set: AS N N O, Frequency: 3



Vertex set: AS C N N, Frequency: 3

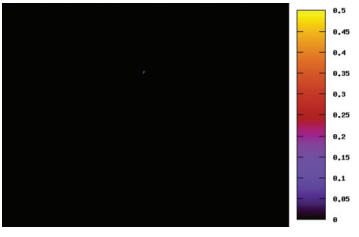


Vertex set: N O S SE, Frequency: 3

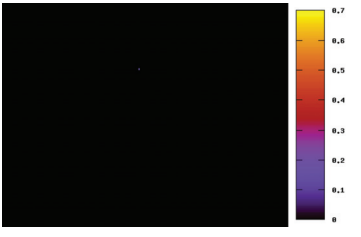


Vertex set: C I O S, Frequency: 2

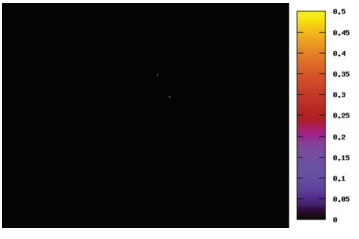
Vertex set: C C F F, Frequency: 2



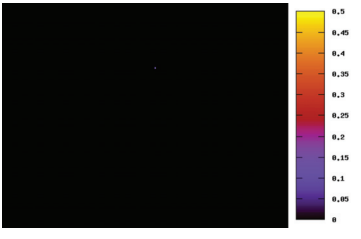
Vertex set: B N O O, Frequency: 2



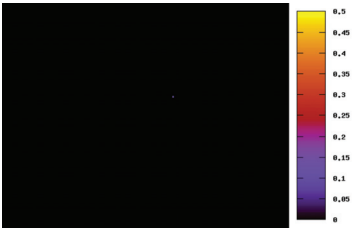
Vertex set: C O P S, Frequency: 2



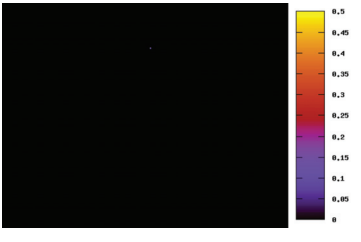
Vertex set: C F O S, Frequency: 2



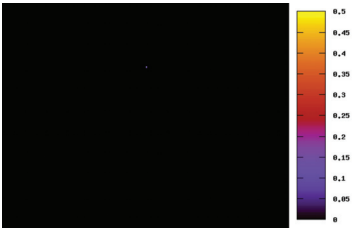
Vertex set: AS C N S, Frequency: 1



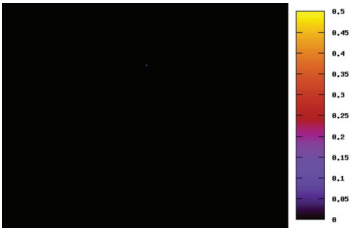
Vertex set: O O S SE, Frequency: 1



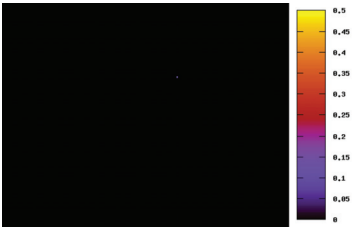
Vertex set: C F F N, Frequency: 1



Vertex set: C C C P, Frequency: 1

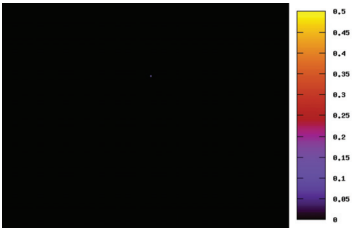


Vertex set: O O P S, Frequency: 1

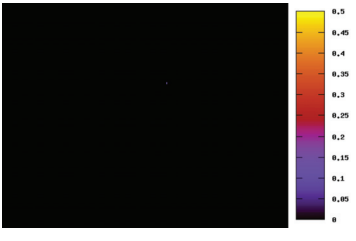


Vertex set: N N S SE, Frequency: 1

Vertex set:: AS O O S, Frequency: 1



Vertex set: AS N O S, Frequency: 1

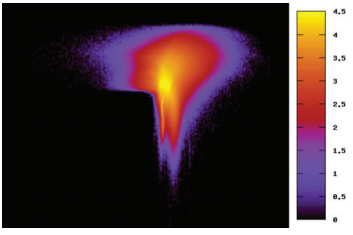


Vertex set: C CL N N, Frequency: 1

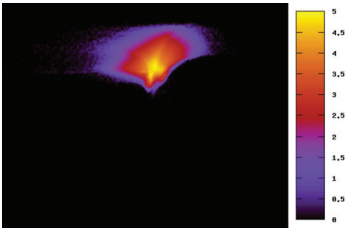
Chapter 6

Appendix 2

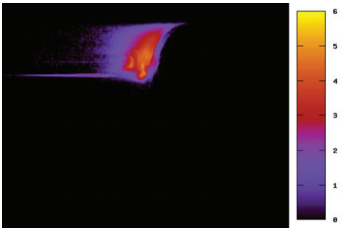
This appendix shows the empirical distribution plots of the volume and tetrahedrality of certain types of tetrahedra as explained on page 47 in section 3.2.2. Tetrahedrality and volume intervals are the same as on 3.6.



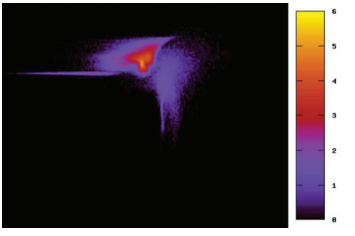
Covalent bond graph: 0000



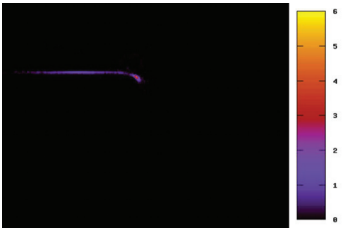
Covalent bond graph: 0011



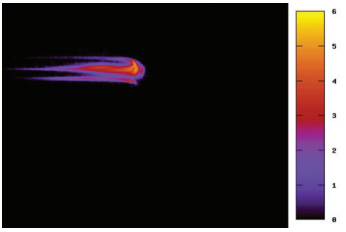
Covalent bond graph: 0112



Covalent bond graph: 1111



Covalent bond graph: 1113

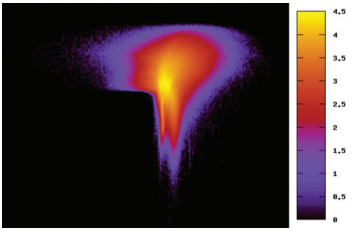


Covalent bond graph: 1122

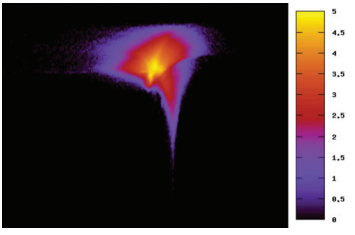
Chapter 7

Appendix 3

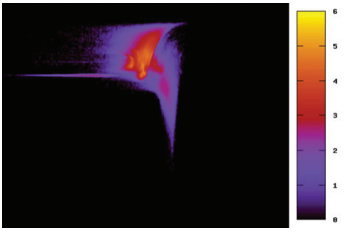
This appendix shows the empirical distribution plots of the volume and tetrahedrality of certain types of tetrahedra as explained on page 47 in section 3.2.2. Tetrahedrality and volume intervals are the same as on 3.6.



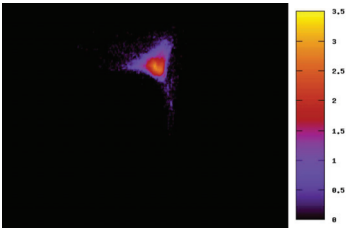
Bond graph: 0000



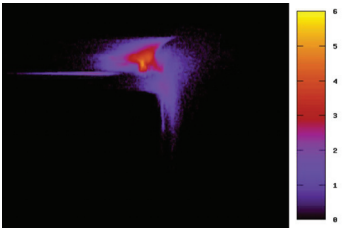
Bond graph: 0011



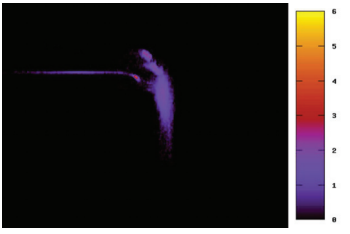
Bond graph: 0112



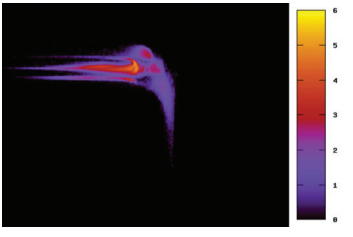
Bond graph: 0222



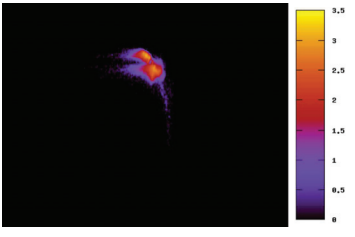
Bond graph: 1111



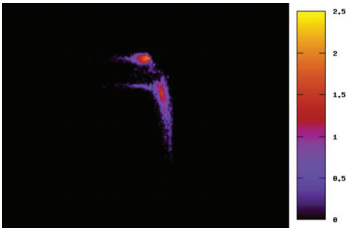
Bond graph: 1113



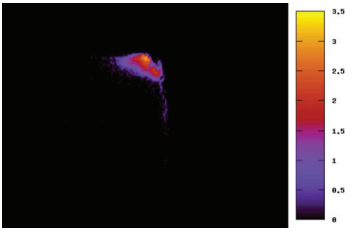
Bond graph: 1122



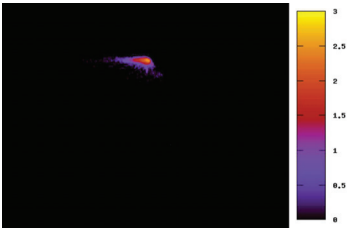
Bond graph: 1223



Bond graph: 2222



Bond graph: 2233



Bond graph: 3333

Acknowledgments

I would like to express my gratitude to my supervisor, Dr. Vince Grolmusz, whose expertise, understanding, and patience, added considerably to my graduate experience. Without him I wouldn't even know what Bioinformatics is. I would like to thank the other members of my research group, the PIT Group, Zoltán Szabadka, Gábor Iván, Dániel Bánky and Árpád Tóth for the assistance they provided at all levels of the research project. They were good colleagues and amazing friends.

I would also like to thank my family for the support they provided me through my entire life and in particular, I must acknowledge my fiancee, Juci, without whose love, encouragement and editing assistance, I would not have finished this thesis in time.

In conclusion, I recognize that this research would not have been possible without the financial assistance of the NKTH research grant sponsoring the TB-INTER project, the Eotvos University (Budapest, Hungary), the Department of Computer Science, and express my gratitude to those agencies.

List of Figures

List of Tables

Bibliography

[1] F. H. Allen, O. Kennard, D. G. Watson, L. Brammer, A. G. Orpen, and R. Taylor. Table of bond lengths determined by x-ray and neutron diffraction. *J. Chem. Soc.*, Perkin Trans. 2:S1–S19, 1987.

[2] H.M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T.N. Bhat, H. Weissig, I.N. Shindyalov, and P.E. Bourne. The protein data bank. *Nucleic Acids Research*, 28:235–242, 2000.

[3] Badry D. Bursulaya, Maxim Totrov, Ruben Abagyan, and Charles L. Brooks. Comparative study of several algorithms for flexible ligand docking. *Journal of Computer-Aided Molecular Design*, 2004.

[4] Vince Grolmusz Dániel Bánky, Rafael Ördög. Nascent: An automatic protein interaction network generation tool for non-model organisms. *Bioinformation*, 3(8):361–363, 2009.

[5] C. Bradford Barber et al. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4):469–483, 1996.

[6] R. K. Singh et al. Delaunay tessellation of proteins: Four body nearest-neighbor propensities of amino acid residues. *Journal of Computational Biology*, 3(2):213–222, 1996.

[7] Skillman AG Kuntz ID. Ewing TJ, Makino S. Dock 4.0: search strategies for automated molecular docking of flexible molecule databases. *J. Comput. Aided. Mol. Des.*, 2001.

[8] F.D.Brown. Voronoi diagramms from convex hulls. *Information processing letters*, 9:223–228, 1978.

[9] Philippe Ferrara, Holger Gohlke, Daniel J. Price, Gerhard Klebe, and Charles L. Brooks. Assessing scoring functions for protein-ligand interactions. *J. Med. Chem*, 2004.

[10] Robert S. Halliday Ruth Huey William E. Hart Richard K. Belew Arthur J. Olson Garrett M. Morris, David S. Goodsell. Automated docking using a Lamarckian genetic algorithm and an empirical binding free energy function. *Journal of Computational Chemistry*, 1998.

[11] Rafael Ördög Vince Grolmusz Gábor Náray-Szabó Gábor Iván, Zoltán Szabadka. Four spatial points that define enzyme families. *Biochemical and Biophysical Research Communications*, 383(4):417–420, 2009.

[12] William Eugene Hart. *Adaptive Global Optimization with Local Search*. PhD thesis, University of California, San Diego, 1994.

[13] G. et al. Ivan. Cysteine and tryptophane anomalies found when scanning all the binding sites in the protein data bank. 2007.

[14] Mark Jerrum and Gregory B. Sorkin. Simulated annealing for graph bisection. In *IEEE Symposium on Foundations of Computer Science*, pages 94–103, 1993.

[15] Brian K. Shoichet John J. Irwin. A free database of commercially available compounds for virtual screening. *J. Chem. Inf. Comput. Sci.*, 2005.

[16] Esther Kellenberger, Jordi Rodrigo, Pascal Muller, and Didier Rognan. Comparative evaluation of eight docking tools for docking and virtual screening accuracy. *Proteins*, 2004.

[17] R. A. Laskowski, M. W. MacArthur, D. S. Moss, and J. M. Thornton. *PROCHECK*: a program to check the stereochemical quality of protein structures. *Journal of Applied Crystallography*, 26(2):283–291, Apr 1993.

[18] D. T. Lee and C. K. Wong. Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees. *Acta Informatica*, 9 (1):23–29, 1977.

[19] Thomas Lengauer. *Bioinformatics - From Genomes to Therapies*. Wiley-VCH, 2007.

[20] Gandhi NR, Moll A, Sturm AW, Pawinski R, Govender T, Lalloo U, Zeller K, Andrews J, and Friedland G. Extensively drug-resistant tuberculosis as a cause of death in patients co-infected with tuberculosis and hiv in a rural area of south africa. *The Lancet*, 368:1575–1580.

[21] Wang R, Fang X, Lu Y, Yang CY, and Wang S. The PDBbind database: methodologies and updates. *J. Med. Chem.*, 48:4111–4119, 2005.

[22] Wang R, Fang X, Lu Y, and Wang S. The PDBbind database: collection of binding affinities for protein-ligand complexes with known three-dimensional structures. *J. Med. Chem.*, 47:2977–2980, 2004.

[23] Vince Grolmusz Rafael Ördög, Zoltán Szabadka. Decomp: A pdb decomposition tool on the web. *Bioinformation*, 3(10):413–414, 2009.

[24] Lengauer T Klebe G. Rarey M, Kramer B. A fast flexible docking method using an incremental construction algorithm. *J. Mol. Biol.*, 1996.

[25] Francisco J. Solis and Roger J.B. Wets. Minimization by random search techniques. *Mathematical Operations Research*, 1981.

[26] F.H. Allen S.R. Hall and I.D. Brown. A new standard archive file for crystallography. *Acta Cryst.*, A47:655–685., 1991.

[27] Zoltan Szabadka and Vince Grolmusz. Building a structured PDB: The RS-PDB database. In *Proceedings of the 28th IEEE EMBS Annual International Conference, New York, NY, Aug. 30-Sept 3, 2006*, pages 5755–5758, 2006.

[28] Martin Stahl Tanja Schulz-Gasch. Binding site characteristics in structure-based virtual screening: evaluation of current docking tools. *Journal of Molecular Modeling*, 2003.

[29] René Thomasen. Flexible ligand docking using evolutionary algorithms. investigating the effects of variation operators and local-search hybrids. *BioSystems*, 2003.

[30] Hartshorn MJ Murray CW Taylor RD. Verdonk ML, Cole JC. Improved protein-ligand docking using gold. *Proteins*, 2003.

[31] Darrell Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 1994.

[32] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997.

[33] Vince Grolmusz Zoltán Szabadka, Rafael Ördög. The ramachandran map of more than 6,500 perfect polypeptide chains. *Biophysical Reviews and Letters*, 2(3):1–5, 2007.

[34] Rafael Ördög. Pydet, a pymol plug-in for visualizing geometric concepts around proteins. *Bioinformation*, 2(8):346–347., 2008.

[35] Rafael Ördög and Vince Grolmusz. Evaluating genetic algorithms in protein-ligand docking. volume 4983 of *Lecture Notes in Computer Science*, pages 402–413. ISBRA: 4-th International Symposium on Bioinformatics Research and Applications, Atlanta, USA, 2008.

[36] Rafael Ördög and Vince Grolmusz. On the bond graphs in the delaunay-tetrahedra of the simplicial decomposition of spatial protein structures. volume 5518 of *Lecture Notes in Computer Science*, pages 1162–1169. IWANN: International Work-Conference on Artificial Neural Networks, Salamanca, Spain, 2009.

[37] Rafael Ördög, Zoltan Szabadka, and Vince Grolmusz. Analyzing the simplicial decomposition of spatial protein structures. *BMC Bioinformatics*, 9(S-11), 2008, InCoB: 6th International Conference on Bioinformatics, Hong Kong.

Publications of Rafael Ördög

Rafael Ördög and Vince Grolmusz. Evaluating genetic algorithms in protein-ligand docking. volume 4983 of *Lecture Notes in Computer Science*, pages 402–413. ISBRA: 4-th International Symposium on Bioinformatics Research and Applications, Atlanta, USA, 2008

Rafael Ördög. Pydet, a pymol plug-in for visualizing geometric concepts around proteins. *Bioinformation*, 2(8):346–347., 2008

Rafael Ördög, Zoltan Szabadka, and Vince Grolmusz. Analyzing the simplicial decomposition of spatial protein structures. *BMC Bioinformatics*, 9(S-11), 2008, InCoB: 6th International Conference on Bioinformatics, Hong Kong

Rafael Ördög and Vince Grolmusz. On the bond graphs in the delaunay-tetrahedra of the simplicial decomposition of spatial protein structures. volume 5518 of *Lecture Notes in Computer Science*, pages 1162–1169. IWANN: International Work-Conference on Artificial Neural Networks, Salamanca, Spain, 2009

Publications not included in the Thesis

Vince Grolmusz Zoltán Szabadka, Rafael Ördög. The ramachandran map of more than 6,500 perfect polypeptide chains. *Biophysical Reviews and Letters*, 2(3):1–5, 2007

Rafael Ördög Vince Grolmusz Gábor Náray-Szabó Gábor Iván, Zoltán Szabadka. Four spatial points that define enzyme families. *Biochemical and Biophysical Research Communications*, 383(4):417–420, 2009

Vince Grolmusz Dániel Bánky, Rafael Ördög. Nascent: An automatic protein interaction network generation tool for non-model organisms. *Bioinformation*, 3(8):361–363, 2009

Vince Grolmusz Rafael Ördög, Zoltán Szabadka. Decomp: A pdb decomposition tool on the web. *Bioinformation*, 3(10):413–414, 2009

Summary of the Thesis

The appearance of computers in the 20th century revolutionized scientific research. Computer-based methods in drug discovery were inevitable. Bioinformatics was born as the study of gene and amino acid sequences, but by now it includes many other fields. Our research focused on *in silico* docking methods, geometrical study of protein structures, and database cleaning. Our earlier articles have been edited and slightly extended to form this thesis.

Chapter 1 outlines our motivations and describes where and how bioinformatics can be useful in drug discovery.

Chapter 2 is dealing with *in silico* docking methods and it is based on the article [35]. First existing docking software are discussed with focus on AutoDock 3 and genetic algorithm with local search. The rest of the chapter introduces the results of the research in the scrIN-SILICO project founded by the European Union. The project's main purpose was to find drug candidates against *Mycobacterium tuberculosis*, but the approach described can be applied to any other bacterial disease. The chapter concludes by marking the limitations of global search as the weakest point of the method, and calls attention to the possibility of utilizing the special - possibly geometrical - properties of the energy function and the specific protein.

Chapter 3 tries to answer the problems stated in the second chapter, by introducing a new way of handling 3D protein structures geometrically with Delaunay decompositions. Although the idea is not entirely new, the field is vastly unexplored, and our current results are only the first steps in exploring the possibilities. In the long term the novel methods, and statistical results presented in [34, 37, 36] may prove useful in protein structure studies, and docking site identification. The chapter is based on these articles, but contains the same statistical results in more detail. Furthermore Appendix 1, Appendix 2 and Appendix 3 contain further supplementary material that has not been published before.

Chapter 4 addresses the problem of data cleaning, which is more related to the field of data mining than to bioinformatics, but its importance in the field is undeniable. Most of the biological data banks were designed in the early stages of informatics, when automated mass processing of entries was not considered. The chapter focuses on the Protein Data Bank (PDB), which contains more than 50,000 entries, and it is a large repository of 3D protein structures used in the third chapter. After briefly discussing classical automated methods of data cleaning in the PDB, a new society-based approach is introduced. WikiPDB is a portal based on the MediaWiki software that has been presented on several conferences. Apart from handling different versions of the same PDB file, it checks for syntax errors. It also provides an API for further error checking plug-ins. The primary aim is to speed up communication between research groups performing automated processing of the data bank, and between specialists of the individual PDBs.

Rövid összefoglaló

A számítógépek megjelenése a XX. században forradalmasította a tudományt, így a számítógépes módszerek megjelenése a gyógyszerkutatásban is elkerülhetetlen volt. A bioinformatika a gén- és aminosav sorrend terén végzett kutatásokkal kezdődött, de napjainkban már sok más tudományterületet is magába foglal. Kutatómunkánk a számítógéppel végzett dokkolási módszerekre, a fehérjeszerkezetek geometriai tanulmányozására és adatbázis tisztításra koncentrált. Jelen tézis alapját korábbi cikkeink szolgáltatják, azonban több még nem publikált eredmény is helyet kapott.

Az 1. Fejezet felvázolja céljainkat, és bemutatja, hol és milyen módon alkalmazható a bioinformatika a gyógyszerkutatásban.

A 2. Fejezet a számítógépes dokkolási módszerekkel foglalkozik, és a [35] cikken alapul. Először tárgyalásra kerülnek a dokkoló szoftverek nagy hangsúlyt fektetve az AutoDock 3-ra és a lokális kereséssel javított genetikus algoritmusokra. A fejezet további részében bemutatjuk az Európai Unió által finanszírozott scrIN-SILICO projekt keretein belül elért eredményeinket. A projekt fő célja a Mycobacterium tuberculosis elleni gyógyszerjelöltek kutatása volt, de megközelítésünk bármely más bakteriális betegség elleni harcban alkalmazható. A fejezet konklúziójaként megemlítjük, hogy a módszer gyenge pontja a globális keresés. Az energiafüggvény és az adott fehérje jellemző - lehetőleg geometriai - tulajdonságainak használatával azonban feltehetőleg javíthatóak ezek az algoritmusok.

A 3. Fejezet az előző részben feltett kérdések megválaszolására tesz kísérletet egy új fajta módszer bevezetésével, mely a Delaunay felbontást használja a 3D fehérjeszerkezetek kezelésére. Habár az ötlet nem teljesen új, ez a terület még felfedezésre vár, jelenlegi eredményeink csupán az első lépések. Hosszútávon ezek a módszerek és az általuk nyert statisztikai adatok hasznosnak bizonyulhatnak a fehérjeszerkezet vizsgálatában és a kötőhelyek azonosításában. A fejezet a [34, 37, 36] cikkekre épül, de az ott felsorolt statisztikai adatokat sokkal részletesebben mutatja be. Ezenkívül a függelékek további kiegészítő anyagokat tartalmaznak, melyeket eddig nem publikáltunk.

A 4. Fejezet az adattisztítás problémakörét ismerteti, ami szorosabban kapcsolódik az adatbányászathoz, de jelentősége a bioinformatikában tagadhatatlan. A biológiai adatbankok többségét a bioinformatika korai szakaszában hozták létre, amikor még nem merült fel az igény nagy mennyiségű adat tömeges feldolgozására. A fejezet a Protein Data Bankra (PDB) koncentrál, amely több, mint 50 000 jó minőségű 3D fehérjeszerkezeteket tartalmaz. Enélkül a hatalmas adatbázis nélkül a 3. fejezetben ismertetett kutatáshoz nem lett volna elegendő adatunk. Miután röviden összefoglalom a PDB klasszikus adattisztítási módszereit, bemutatok egy új, közösség alapú megközelítést.

A WikiPDB egy MediaWiki alapú portál. Azonkívül, hogy ugyanazon PDB fájl több verzióját is kezeli, szintaktikai hibákat is keres. Továbbá egy API-t is biztosít további hibakereső bővítmények számára.